TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

VALTTERI LUOMA
PILVIPALVELUN TOTEUTUS TYÖKONEIDEN AJONAIKAISEEN ETÄSEURANTAAN

Diplomityö

Tarkastajat: Yliop. leht. Heikki Huttunen ja prof. Hannu-Matti Järvinen
Tarkastajat ja aihe hyväksytty
Tieto- ja sähkötekniikan tiedekunnan tiedekuntaneuvoston
kokouksessa 7.12.2016

TIIVISTELMÄ

VALTTERI LUOMA: Pilvipalvelun toteutus työkoneiden ajonaikaiseen etäseurantaan
Tampereen teknillinen yliopisto
Diplomityö, 51 sivua
maaliskuu 2017
Tietotekniikan koulutusohjelma
Pääaine: Data Engineering
Tarkastajat: Yliop. leht. Heikki Huttunen ja prof. Hannu-Matti Järvinen
Avainsanat: Esineiden internet, pilvipalvelu, etäseuranta, JavaScript

Polttomoottorikäyttöisiä työkoneita, kuten trukkeja ja traktoreita, huolletaan tietyn käyttötuntimäärän jälkeen. Tavallisesti työkoneita huoltava yritys joutuu manuaalisesti lukemaan käyttötuntien määrän jokaisesta laitteesta. Kyseinen toimintatapa ei ole kustannustehokas. Tässä työssä esitellään, miten Intopalo Oy:n asiakkaalle toteutettiin työkoneiden käyttötuntien etäseurantajärjestelmä.

Projektissa toteutettiin pilvipalvelu, johon työkoneisiin kiinnitetyt etälaitteet raportoivat kertyneet käyttötunnit. Työkoneet liitettäisiin siis esineiden internetiin (IoT). Projektiin kuului myös etälaitteiden valinta ja muutaman testilaitteen kokoaminen.

Web-palvelun totetuttamisessa käytettiin MEAN-arkkitehtuuria, joka koostuu MongoDB NoSQL-tietokannasta, Express.js- ja AngularJS-ohjelmistokehyksestä sekä Node.js-palvelinohjelmistosta. Arkkitehtuurin kaikkia komponentteja pystytään kehittämään JavaScript-ohjelmointikielellä. Palvelun ylläpitämisessä ja tietojen tallentamisessa hyödynnettiin Amazon Web Servicesin pilvipalveluja.

Valittu etälaite oli Particle Electron, joka on kaupallinen kehitysalusta integroidulla 2G-modeemilla esineiden internetin laitteiden rakentamiseen. Laite muistuttaa Arduinoa, joka on toinen yleinen kehitysalusta. Particle Electron vaati valmistajan oman pilvipalvelun käyttöä, jos viestit haluttiin lähettää salattuina.

Toteutettu pilvipalvelu osoittautui toimivaksi digitaaliseksi huoltokirjaksi, johon huoltomiehet pystyivät merkitsemään tehtyjä toimenpiteitä eri työkoneille. Etälaitteiden lähettämän käyttötuntidatan avulla pystyttiin arvioimaan, milloin työkone täytyy seuraavan kerran huoltaa. Toteutetut etälaitteiden testiversiot osoittautuivat luotettaviksi. Niiden lähettämät viestit saapuivat lähes aina perille. Toteutettu palvelu osoittautui siis kokonaisuudessa toimivaksi ratkaisuksi.

ABSTRACT

VALTTERI LUOMA: Cloud service implementation for remote monitoring of machines
Tampere University of Technology
Master of Science Thesis, 51 pages
March 2017
Master's Degree Programme in Information Technology
Major: Data Engineering
Examiner: Univ. lect. Heikki Huttunen and prof. Hannu-Matti Järvinen
Keywords: internet of things, cloud service, remote monitoring, JavaScript

Combustion engine powered machines, like forklifts and tractors, need maintenances based on the amount of usage hours. Company that performed these maintenances on work machines had to manually read the hours from the machines. In this work, it is presented how machine remote monitoring system was implemented for Intopalo Oy's client.

The project included implementation of cloud service. Remote monitoring devices reported the elapsed usage hours from the work machines to the service. This means that the machines were connected to the internet of things. Project included also choosing the remote device and assembly of a few demo devices.

MEAN architecture was used in the cloud service implementation. It consists of MongoDB NoSQL database, Express.js framework, AngularJS framwork and Node.js server. All parts of the architeture can be developed with JavaScript programming language. The service was built on Amazon Web Services cloud platform.

The chosen remote device was Particle Electron. It is a commercial development board with integrated 2G modem. It resembles a lot Arduino, which is another, very common development board. Particle Electron required the use of Particle's own cloud service for sending encrypted messages.

Implemented cloud service proved to be usable digital service manual for the machines. Mechanicians were able to log completed maintenaces and repairs for each machine. Date of the next maintenace for a machine could be estimated by analyzing the usage hour data that was sent by the remote devices. The implemented demo versions of the remote device proved out to be reliable. Messages that they sent, almost invariably arrived to the cloud service. The implemented service as a whole proved out to be a well working solution.

ALKUSANAT

Tässä se taitaa olla.

Elikkäs. Kiitokset Juuso Partaselle, joka luki ja kommentoi dokumentoitua ajatuksenvirtaani ja auttoi pohtimaan erilaisia näkökulmia työn kannalta. Kiitokset Anetelle, joka jaksoi minua laittaessaan pilkkuja ja pisteitä paikoilleen. Kiitokset myös muulle perheelle tuesta ja turvasta opiskelujen aikana.

Intopaloa haluan kiittää tämän diplomityön mahdollistamisesta. Kesäprojekti, jonka aikana pääsi oppimaan valtavasti uutta, johti ripeään diplomityön kasaamiseen. Intopalon työntekijöistä haluan erityisesti kiittää Juanjo Diazta, joka opasti minut web- ja JavaScript-kehityksen ihmeellisiin maailmoihin ja Juha Vuolletta, joka toimi projektin varsinaisena vetäjänä.

Tampereen teknilliselle yliopistolle kiitokset loistavista puitteista opiskelulle. Kiitokset myös niille luennoitsijoille, jotka rakastavat juuri sitä, mitä tekevät. Heidän asenteensa piti opiskelumotivaatiota yllä. Erityiskiitokset työtä ohjanneille Heikki Huttuselle ja Hannu-Matti Järviselle, jotka kuuluvat myös edellämainittuihin luennoitsijoihin.

Do or do not. There is no try.

Yoda, *Episode V: The Empire Strikes Back*

Tampereella, 28.3.2017

Valtteri Luoma

SISÄLLYS

LYHENTEET JA MERKINNÄT

IoT	Internet of Things
REST	Representational State Transfer
MVC	Model-View-Controller
JSON	JavaScript Object Notation
SQL	Structured Query Language
NoSQL	Not only SQL
XML	Extended Markup Language
HTTP	Hypertext Transfer Protocol
HTML	Hypertext Markup Language
URL	Uniform Resource Locator
API	Application Programming Interface
SaaS	Software as a Service
PaaS	Platform as a Service
IaaS	Infrastructure as a Service

1. JOHDANTO

Esineiden internet (Internet of Things) on ollut viime vuosien nouseva trendi. Useat yritykset ovat esitelleet alustoja, laitteita, verkkoja ja verkkoteknologioita esineiden liittämiseksi internetiin. Esineiden internetin uskotaan tarjoavan tulevaisuudessa valtavasti mahdollisuuksia kehittää uudenlaisia palveluita ja sovelluksia. Esineiden internetin tarjoamia mahdollisuuksia ei tosin osata vielä hyödyntää: Gartnerin mukaan [41] esineiden internet on pidempiaikainen hypetyksen aihe, mutta vain harvat ratkaisut tai tuotekategoriat ovat lyöneet itsensä läpi. Tässä työssä pyritään kehittämään esineiden internetin sovellus, josta on hyötyä useille eri käyttäjäryhmille. Samalla saadaan näkemys siitä, millaisia ongelmia tai haasteita esineiden internetin ratkaisut kohtaavat.

Työkoneet, kuten trukit, traktorit ja maansiirtokoneet, on huollettava yleensä tietyn käyttötuntimäärän jälkeen. Varsinkin työkoneet, joiden voimanlähteenä on polttomoottori. Käyttötuntien määrä selviää moottorin käyttötunteja laskevasta käyttötuntimittarista. Vanhemmissa työkoneissa käyttötuntimittari on analoginen, joka on luettava manuaalisesti. Monissa uudemmissa työkoneissa on valmistajakohtaisia digitaalisia järjestelmiä, joiden avulla käyttötunnit pystytään lukemaan etäyhteyden avulla. Yhteistä standardia eri valmistajien työkoneilla ei kuitenkaan ole. Tällöin yritykset, joilla on useita erityyppisiä ja eri-ikäisiä työkoneita, eivät pysty tarkkailemaan kaikkien työkoneiden käyttötuntimääriä lukematta tunteja manuaalisesti työkoneesta. Käyttötuntien manuaalinen lukeminen ei ole kustannustehokas ratkaisu. Jos huoltomiesten tehtäviin kuuluu selvittää asiakkaiden työkoneiden käyttötuntien määriä, työaikaa kuluu tällöin ei-tuottavaan työhön.

Tässä työssä kehitettävä järjestelmä tarjoaa käyttötuntidataa asiakkaan saataville etäyhteyden avulla. Käyttötunnit kerätään jokaiseen työkoneeseen kiinnitettävän etälaitteen avulla. Sama järjestelmä pystyisi hallinnoimaan työkoneille tehtäviä huoltoja ja korjauksia. Huoltojen merkitseminen onkin käyttötuntien kannalta olennaista, jotta tiedetään, milloin huollot on suoritettava.

Luvussa 2 käydään läpi projektin rakennetta ja vaiheita. Luvussa 3 tutkitaan webpalvelun toteuttamiseen valittuja suunnittelumalleja ja tekniikoita, näille valittuja

ratkaisuja ja itse web-palvelun toteutusta. Luvussa 5 käsitellään etälaitteen vaatimuksia, mahdollisia vaihtoehtoja, valittua ratkaisua ja lopullista toteutusta. Luvussa 6 käsitellään projektin toteuttamisesta saadut johtopäätelmät, sekä kuvaillaan mahdollisia jatkokehitysmahdollisuuksia.

2. LÄHTÖKOHDAT

Tässä luvussa kerrotaan, mitä esineiden internet tarkoittaa ja mitä ovat esineiden internetin laitteet. Luvussa kuvataan myös projektin lähtökohtana oleva ongelma. Tämän jälkeen kuvaillaan, millaisena projektina ongelma ratkaistiin ja mitä vaiheita se sisälsi.

2.1 Internet of Things

Internet of Things (IoT), eli esineiden internet, on ajatus maailmanlaajuisesta infrastruktuurista, jossa fyysiset esineet pystyvät kommunikoimaan keskenään. Esineet ovat niin sanotusti "älykkäitä". Toimintalogiikan ja antureiden avulla ne ymmärtävät ympäristöään ja reagoivat ympäristön tapahtumien mukaan, esimerkiksi muiden laitteiden ja laitteiden käyttäjien kanssa. Esineiden internetin laitteet, esineet ja asiat voivat olla hyvin erilaisia. Jotkin laitteet voivat olla yhtä ohjelmoitavia ja monipuolisia kuin tietokoneet. Osa on vain ohjelmoitavalla RFID-tunnisteella [23] esineiden internetiin liitetty, "normaali" esine. [36]

Hyvä esimerkki "älykkäästä" kuluttajatuotteesta, esineiden internetiin kytketystä laitteesta, on Amazonin Echo. Echo on kaiutin, joka on yhdistetty internetiin ja joka kuuntelee ja ymmärtää käyttäjän puhetta. Puheesta se pyrkii havaitsemaan komentoja, joita se on ohjelmoitu toteuttamaan. Näitä toimintoja ovat esimerkiksi sääennusteiden kertominen ja tuotteiden ostaminen Amazonin verkkokaupasta. Se pystyy myös ohjaamaan muita "älykkäitä" esineiden internetin laitteita. [3]

Gartnerin laatiman esineiden internetin hypekäyrän mukaan suurin osa esineiden internetiin liittyvistä ratkaisuista on ollut useamman vuoden lähellä hypetyksen huippua. Hypetyksen laantuminen ja esineiden internetin ratkaisujen kehittyminen tuottaviksi ratkaisuiksi kestää arvioiden mukaan vielä useita vuosia. Tällä hetkellä hypetettyjä ratkaisuja ovat muun muassa edulliset kehitysalustat, kuten Raspberry Pi [20], IoT-alustat, IoT-arkkitehtuurit, koneoppiminen ja asiakaspalvelun IoT. [41] Esineiden internet on siis tällä hetkellä kuuma puheenaihe, mutta konseptina vielä alkutekijöissään. Sen tuotteet ja tuotekategoriat hakevat vielä muotoaan.

2.2 Ratkaistava ongelma

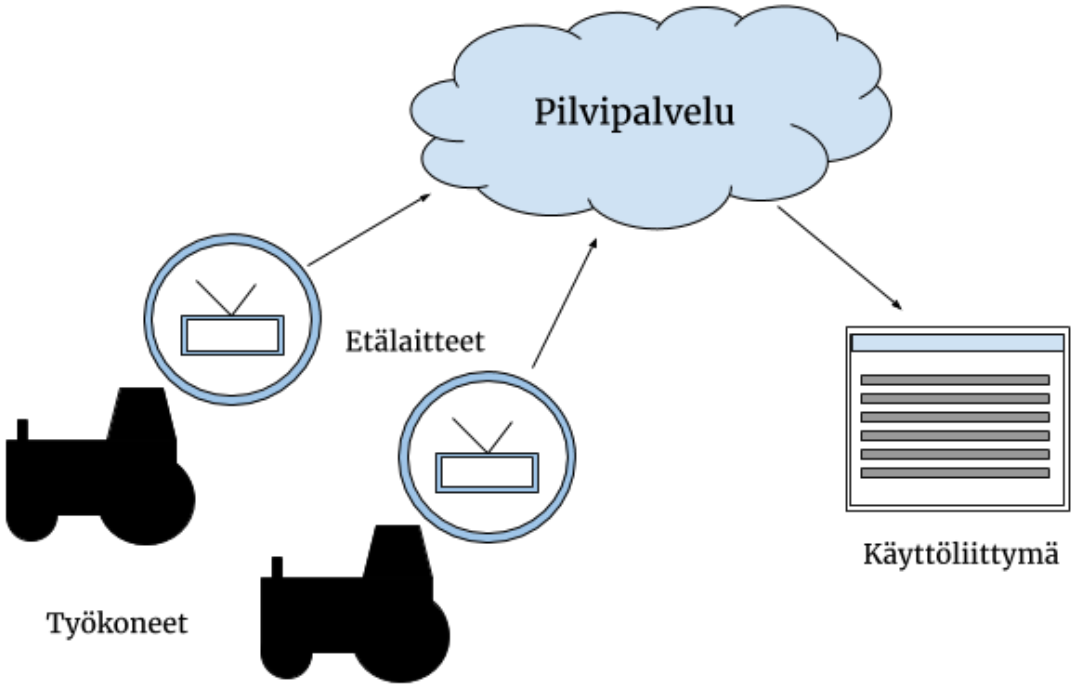
Projekti toteutettiin ohjelmistoyritys Intopalo Oy:n asiakkaalle. Asiakas oli työkoneita huoltava keskisuuri yritys. Huollettavat koneet sijaitsivat etäällä toisistaan, useissa eri asiakasyrityksissä. Koneiden käyttö oli vaihtelevaa, mikä tuotti haasteita huoltojen aikatauluttamiselle. Polttomoottorilla toimiviin koneisiin huollot tehtiin käyttötuntien perusteella, esimerkiksi kolmensadan tunnin välein. Sähkökäyttöisiä koneita huollettiin yleensä vuoden tai joidenkin kuukausien välein käyttötunneista riippumatta. Huoltojen aikataulutukseen liittyi myös varaosien tilausta. Ennakoimalla lähitulevaisuudessa tarvittavat varaosat, asiakas pystyisi säästämään merkittäviä summia rahtikustannuksista. Säästöä syntyisi, jos varaosat voitaisiin tilata toimitettavaksi hitaamman, meriteitse tapahtuvan kuljetuksen avulla. Lähtötilanteessa asiakas joutui kuitenkin usein tilaamaan varaosat nopealla aikataululla lentorahdin avulla, koska huoltoja ei pystytä ennakoimaan tarpeeksi hyvin. Lentorahdin toimituskulut ovat yleensä meritiekuljetukseen verrattuna huomattavasti suuremmat.

Heti projektin alussa oli siis selvästi nähtävissä asiakkaan motivaatio saada tietoonsa työkoneiden käyttötunnit ilman, että kenenkään tarvitsisi lukea niitä manuaalisesti. Huollettava kalusto oli vaihtelevaa niin iältään kuin malleiltaan. Koneista ei tunnistettu yhtenäistä tapaa käyttötuntien keräämiseen suoraan työkoneiden järjestelmistä. Käyttötuntien seurannan mahdollistamiseksi oli vaihtelevaan työkonekantaan kiinnitettävä etälaiteet, jotka pystyisivät raportoimaan koneiden käyttötunteja riippumatta koneen tyypistä tai mallista.

Etälaitteen korkean tason toimintalogiikka oli siis jo projektin alussa tiedossa. Kun koneen moottori olisi käynnissä, etälaite saisi virtaa ja alkaisi laskemaan käyttötunteja. Käyttötunnit raportoitaisiin internetin välityksellä eteenpäin. Jotta etälaitteet pystyisivät raportoimaan käyttötunteja, oli niitä varten rakennettava tai valittava web-palvelu, johon kyseistä dataa voitaisiin lähettää. Data kulkee työkoneisiin kiinnitetyistä etälaitteista pilvipalveluun ja on sieltä nähtävissä web-käyttöliittymän avulla. Kuvassa 2.1 on nähtävissä järjestelmän korkean tason arkkitehtuuri. Korkealla tasolla arkkitehtuuri on yksinkertainen. Etälaitteet lähettävät viestejä pilvipalveluun, jota hallinnoidaan käyttöliittymän avulla.

2.3 Projektin vaiheet

Projekti toteutettiin kahdessa vaiheessa. Ensimmäinen vaihe oli selvitystä ja tutkimusta varten. Tämän aikana valittiin mahdollisesti toteutettavan järjestelmän osat ja teknologiat. Selvitysvaihe kesti noin kaksi viikkoa, jonka aikana valittiin web-palvelun toteutustapa sekä kartoitettiin parhaat etälaitevaihtoehdot. Onnistuneen



Kuva 2.1 *Suunnitellun järjestelmän korkean tason arkkitehtuuri. Data lähetetään työkoneisiin kiinnitetyistä etälaitteista pilivpalveluun ja on sitä kautta nähtävissä web-palvelun käyttöliittymän kautta.*

selvitysvaiheen jälkeen seurasi toteutusvaihe, jonka kesto oli noin kolme kuukautta. Toteutusvaiheen aikana rakennettiin suunniteltu järjestelmä siihen asti, että se voitiin ottaa käyttöön huoltoyrityksen sisäisessä käytössä. Etälaitteen testiversioita rakennettiin kenttätestausta varten kaksi kappaletta.

2.3.1 Selvitysvaihe

Selvitysvaiheen tarkoituksena oli löytää ratkaisu projektin alussa kuvattuun ongelmaan. Selvitysvaihe oli kestoltaan noin kaksi viikkoa työtunteina, mutta jakautui pidemmälle aikavälille. Selvitysvaiheen jälkeen oli tiedossa, miten web-palvelu toteutettaisiin ja mitä eri vaihtoehtoja valittavaksi etälaitteeksi olisi.

Selvitysvaiheen aikana kuunneltiin asiakkaan mielipiteitä erilaisista ratkaisuvaihtoehdoista ja tulevan palvelun käyttökontekstista. Asiakkaalle esiteltiin esimerkiksi web-palvelun käyttöliittymän paperiprototyyppejä, jotta toteutettavan web-palvelun rakenne ja ulkonäkö olisi selvä kaikille osapuolille. Asiakkaalla oli selkeä vaatimus "liikennevaloista" palvelun etusivulla. Niistä näkisi, mikä työkone olisi huollettava seuraavaksi. Palavereissa selvitettiin myös tarkempia vaatimuksia etälaitteelle, kuten

minkälaisia käyttöjännitteitä työkoneista oli saatavilla. Työkoneiden sähköjärjestelmät olivat pääasiassa 12, 24 ja 48 voltilla toimivia. Usein työkoneista löytyi useampia jännitteitä, eikä työkoneita huoltavalla henkilöstöllä ollut tarkkaa tietoa, missä johdossa olisi mitäkin jännitettä. Etälaite tulisi siis tarvitsemaan jännitemuuntimen, joka muuntaisi minkä tahansa edellä mainituista jännitteistä etälaitteelle sopivaksi. Etälaite saattaisi myös joutua olemaan ulkotiloissa, joten sen tulisi kestää kosteutta ja muita sääoloja, mikä asetti vaatimuksia etälaitteen koteloinnille.

Selvitysvaiheen lopuksi asiakas teki päätöksen web-palvelun toteuttamisesta sekä valitsi etälaitteen kahdesta mahdollisesta laitevaihtoehdosta. Valittu web-palvelun toteutusarkkitehtuuri oli Intopalon esittämä suositus, joka perustui aikaisempaan osaamiseen kyseisten tekniikoiden kanssa. Etälaitteen valinnassa asiakkaalle esitettiin laitevaihtoehtojen hyödyt ja haitat, jolloin asiakas sai valita etälaitteen kyseisten tietojen pohjalta.

2.3.2 Toteutusvaihe

Toteutusvaihe oli selvitysvaihetta pitkäkestoisempi, noin kolmen kuukauden pituinen ajanjakso. Vaiheeseen siirryttäessä tiedossa oli toteutettavan web-palvelun pääasiallinen rakenne ja ulkonäkö. Tiedossa oli myös etälaite, josta lähdettiin kehittämään työkoneeseen sopivaa käyttötuntien raportoijaa. Toteutusvaiheen aikana asiakaalta kyseltiin ja saatiin palautetta eri ominaisuuksien toteutuksesta. Tällöin asiakas näki, miten joku ominaisuus käytännössä toimi ja osasi antaa palautetta siitä, oliko ominaisuus hyvä vai ei. Ominaisuuksia päivitettiin saadun palautteen perusteella.

Toteutusvaiheen aluksi tehtiin MVP-versio (Minimum Viable Product) web-palvelusta. MVP kuvaa tuotteen kehitysvaihetta, jossa sillä alkaa olemaan jotain arvoa tuotteen käyttäjälle. Tiedettiin, että vaatimukset varsinkin web-palvelun ulkonäöstä tulevat jossain määrin muuttumaan asiakkaan nähtyä varsinaista web-palvelua. Tällaista iterointia suoritettiin muutamia kertoja. Asiakkaan toiveet web-palvelusta olivat pääasiassa käyttöliittymän ulkonäön muokkaamista ja pienien toiminnallisuuksien lisäämistä.

Etälaitteen kehittäminen oli hitaampaa kuin web-palvelun, koska se sisälsi erilaisten osien etsimistä ja tilaamista. Varsinaisen etälaitteen ohjelmiston kehittäminen oli suoraviivaista ohjelmiston yksinkertaisuuden takia. Aikaa kuluikin enemmän jännitemuuntimien ja virtalähteen etsintään. Jännitemuuntimet muuntaisivat korkeammat jännitteet etälaitteelle sopivaksi ja virtalähteen avulla testattaisiin jännitemuuntimien toimivuutta.

2.4 Web-palvelun kuvaus

Ennen palvelun suunnittelun aloittamista asiakas kuvaili, mitä web-palvelulla pitäisi pystyä tekemään ja millaisia rooleja palvelun käyttäjillä olisi. Kuvailun avulla kerättiin seuraavanlaiset vaatimukset:

- Web-palvelun tulee toimia digitaalisena huoltokirjana asiakkaiden kaikille työkoneille riippumatta siitä, onko niissä kiinni etälaitetta.
- Palvelua käyttävät ylläpitäjät, huoltomiehet ja asiakkaat.
- Ylläpitäjä hallinnoi palvelussa asiakkaita, koneita, etälaitteita ja palvelun käyttäjätunnuksia.
- Huoltomies merkitsee koneille tehdyt huollot ja korjaukset palveluun, jossa ne ovat asiakkaan nähtävissä.
- Web-palvelun tulee automaattisesti päivittää etälaitteilta tulevat käyttötunnit koneiden tietoihin.
- Huoltomerkinnän tulee noudattaa koneelle laadittua huolto-ohjelmaa myös web-palvelussa.
- Palveluun pitää pystyä tallentamaan työkoneiden huolto-ohjeita. Ne täytyy myös pystyä lataamaan palvelusta huollon tekemistä varten.
- Käyttöliittymän tulee olla selkeä, eikä se saisi sisältää ylimääräisiä elementtejä.

Näiden vaatimusten avulla lähdettiin hahmottelemaan toteutettavaa web-palvelua. Vaatimukset oli suurpiirteisiä, joten asiakkaan mielipiteitä kuunneltiin ja kysyttiin eri ominaisuuksien toteutuksen aikana.

3. PALVELUN TEKNIIKAT

Tässä luvussa kuvataan projektiin liittyvän web-palvelun toteutukseen tarvitut suunnittelumallit ja vaaditut tekniikat. Toteutettu web-palvelu oli web-sovellus. Web-sovellus määritellään luvussa 3.2.

Kun web-palvelun vaatimuksia tarkasteltiin, huomattiin, että sen vaatimukset saattaisivat muuttua projektin aikana. Asiakas saattaisi huomata jonkin ominaisuuden olevan epäkäytännöllinen, tai erilainen kuin oli kuvitellut. Tällöin tarkoitusta varten ei kannattanut valita valmista web-palvelua, vaan toteuttaa palvelu itse. Valmiin web-palvelun muokkaaminen asiakkaan tarpeisiin olisi todennäköisesti mahdotonta. Itse kehitettävä web-palvelu mahdollisti joustavan lähetymistavan projektin toteuttamiseen. Toteutusvaiheessa voitiin siirtyä suoraan selvitysvaiheessa tehtyjen vaatimusten toteuttamiseen.

Web-palvelua suunniteltiin ylläpidettävän pilvipalvelutarjoajan pilvipalvelussa, esimerkiksi Amazon Web Service:ssä. Etälaitteet lähettäisivät 2G-verkon välityksellä dataa web-palveluun, joka käsittelisi sen ja päivittäisi käyttäjien saataville. Rajapintasuunnittelussa noudatettiin Representational State Transfer - eli REST-arkkitehtuurimallia, joka on arkkitehtuurinen malli hajautetuille hypermediajärjestelmille. [34]

3.1 Suunnittelumallit

Luvussa kuvataan projektin toteutukseen käytettyjä suunnittelumalleja. Suunnittelumalli on uudelleenkäytettävä ratkaisu ongelmaan, joka vaatii suunnittelutyötä. Suunnittelumalli on kuvaus yleisestä ongelmasta ja kyseisen ongelman ratkaisun ytimestä. Se kuvaa ratkaisun vain korkealla tasolla, jolloin yksikään samaa suunnittelumallia käyttävä toteutus ei ole samanlainen. Esimerkiksi ohjelmistoarkkitehtuurit ovat suunnittelumalleja. [30] Luvussa kuvattavat asiakas-palvelinarkkitehtuuri, REST- ja Model-View-Controller-arkkitehtuuri olivat kaikki ohjaamassa toteutetun web-sovelluksen rakennetta.

3.1.1 Asiakas-palvelin -arkkitehtuuri

Asiakas-palvelinarkkitehtuuri on web-sovellusten yleisin arkkitehtuurinen malli. Palvelin kuuntelee asiakkaiden kutsuja. Asiakas on usein selain, joka haluaa käyttää palvelimen palveluita. Asiakas kutsuu palvelimen tarjoamia palveluita. Palvelin joko hyväksyy tai hylkää kutsun vastaamalla asiakkaalle. Palvelin on yleensä päättymätön prosessi, joka tarjoaa palveluitaan usein useammalle asiakkaalle samanaikaisesti.

Asiakas-palvelin-arkkitehtuurin tarkoituksena on yksinkertaistaa palvelimen rakennetta, jotta palvelun skaalaaminen voitaisiin toteuttaa yksinkertaisesti. Tällöin käyttöliittymäkomponentit eriytyvät yleensä asiakaspuolen ohjelmistoon. Rakenne mahdollistaa myös molempien osien kehittämisen erikseen, kunhan näiden välinen rajapinta ei muutu. Asiakas-palvelinmalli ei itsessään rajoita sitä, miten ohjelmiston tila tulisi jakaa asiakkaan ja palvelimen välillä. [34, aliluku 3.4.1]

3.1.2 Representional State Transfer

Representional State Transfer eli REST on Fieldingin väitöskirjassaan [34] esittelemä arkkitehtuurinen malli hajautetuille hypermediajärjestelmille, eli esimerkiksi web-palveluille. REST ei keskity sen komponenttien toteutukseen tai protokollien syntaksiin, vaan eri komponenttien rooleihin ja arkkitehtuurin käyttämiseen websovelluksissa. REST hallitsee dataelementtejä siirtämällä informaatiota sen talletuspaikasta sen käyttöpaikkaan, esimerkiksi ihmiskäyttäjälle luettavaksi. Sen tapa siirtää informaatiota perustuu yhteiseen ymmärrykseen datan tyypeistä, sekä rajapinnan rajoitetusta näkyvyydestä.

REST koostuu kuudesta arkkitehtuaalisesta rajoitteesta. Nämä ovat:

1. Asiakas-palvelin -malli, joka kuvattiin aliluvussa 3.1.1.
2. Tilattomuus, eli palvelin ei tallenna tai tiedä asiakkaan tilaa.
3. Välimuistin käyttö (cache). Osa palvelimen vastauksista voidaan tallettaa välimuistiin. Tämä mahdollistaa palvelun tehokkaamman skaalautumisen.
4. Yhtenäinen rajapinta, joka yksinkertaistaa arkkitehtuuria.
5. Kerrosrakenne, joka myös tehostaa palvelun skaalaamista ja mahdollistaa tietoturvaominaisuuksia.

6. Koodi vaadittaessa -periaate (Code-on-demand), joka mahdollistaa palvelimen lähettävän asiakkaalle suoritettavaa koodia, joka muokkaa asiakkaan toimintaa hetkellisesti.

REST-komponentit viestivät lähettämällä resurssien esitysmuotoja tietyssä formaatissa. Resurssi on informaation abstraktio. Konferenssijulkaisu tai lähdekoodi ovat esimerkkejä resursseista. Useampi resurssi voi osoittaa samaan informaatioon. Esimerkiksi lähdekoodin viimeisin versio ja versio 1.23 voivat tarkoittaa samaa asiaa, mutta ne ovat erilliset resurssit. Tällöin linkkejä resurssiin ei tarvitse muuttaa, vaikka esitysmuoto taustalla vaihtuisi. Esimerkiksi lähdekoodin viimeisimpään versioon osoittava linkki voisi pysyä samana.

REST käyttää tunnisteita resurssien identifioimiseen. Tunnisteen valitseminen jätetään arkkitehtuurin laatijalle. REST luottaa laatijan valitsevan parhaiten sopivan tunnisteen kullekin resurssille. REST siirtää resurssin esitysmuodon komponentilta toiselle. Esitysmuoto on tietty määrä tavuja ja metadataa siitä, miten kyseinen data tulee tulkita. Resurssin esitysmuoto voi olla HTML-dokumentti tai kuva, jonka identifioi esimerkiksi URL-osoite. [34, luku 5]

Web-palvelun rajapintaa, joka noudattaa REST-arkkitehtuuria, kutsutaan *RESTful API*:ksi. API, eli *Application Programming Interface*, tarkoittaa ohjelmointirajapintaa. Kun tämäntyyppinen rajapinta käyttää HTTP-protokollaa, hyödyntää se URL-osoitteen lisäksi HTTP:n yleisimmin käytettyjä standardimetodeja. Standardimetodit ovat verbejä, joihin kuuluu muun muassa GET, PUT, POST, PATCH ja DELETE. GET-metodin avulla luetaan tietoa, eli resursseja. Resurssi korvataan kokonaan uudella PUT-metodin avulla. PATCH-metodia käytetään jonkin resurssin päivittämiseen, kun resurssia ei haluta korvata kokonaan uudella, vaan vain osa resurssista. POST-metodin avulla luodaan kokonaan uusi resurssi. DELETE-metodilla poistetaan olemassaoleva resurssi. Taulukosta 3.1 löytyvät esimerkit HTTP-metodien käytöstä REST-rajapinnan kanssa. [22, 27]

Kuten esimerkeistä huomataan, rajapinnan käyttö on hyvin intuitiivista. Verbit määrittelevät, mikä toiminta millekkin resurssille halutaan tehdä.

3.1.3 Model-View-Controller -arkkitehtuuri

Model-View-Controller -arkkitehtuuri, eli MVC-arkkitehtuuri, on interaktiivisia sovelluksia varten kehitetty suunnittelumalli. Siinä erotetaan käyttöliittymä alla olevasta datasta, jota käyttöliittymän avulla esitetään. Mallissa on kolme osaa: Model,

Taulukko 3.1 *Esimerkit HTTP-metodien käytöstä*

HTTP verbi	Toiminta	URL	Toiminta
GET	Lue	www.esim.com/cars	Palauttaa kaikki autot
GET	Lue	www.esim.com/cars/123	Palauttaa auton, jonka ID on 123, tiedot.
PUT	Päivitä/ korvaa	www.esim.com/cars/123	Päivittäää tai korvaa auton, jonka ID on 123, tietoja
POST	Luo	www.esim.com/cars	Lisää uuden auton
PATCH	Päivitä/ muokkaa	www.esim.com/cars/123	Päivittää auton, jonka ID on 123, tietoja
DELETE	Poista	www.esim.com/cars/123	Poistaa auton, jonka ID on 123

View ja Controller. Viewin tehtävä on esittää informaatiota käyttäjälle ja Controllerin kanssa se prosessoi käyttäjän interaktiot. Model sisältää esitettävän informaation ja logiikan, jonka avulla tätä informaatiota muokataan käyttäjän vuorovaikutuksen mukaan. MVC helpottaa sovelluksen kehitystä ja ylläpitoa, koska sovelluksen ulkonäkö voi muuttua huomattavasti ilman, että sovelluksen logiikkaa muutetaan.

MVC-arkkitehtuurin toteuttaminen web-sovelluksissa ei ole ongelmatonta, koska web-sovellus täytyy jakaa asiakkaalle ja palvelimelle. Kehittäjän täytyy siis tehdä valintoja esimerkiksi tekniikoiden suhteen, jotka määrittelevät, miten sovelluksen eri osat tullaan jakamaan palvelimen ja asiakkaan välille. Asiakaspuolen ohjelmisto voi olla kevyempi, jolloin se toteuttaa käytännössä vain View-osaa MVC-mallista. Tällöin Controller ja Model sijaitsevat palvelimella. Jos asiakaspuoli on suurempi, Controller ja Model voivat sijaita myös asiakkaan puolella. MVC-arkkitehtuuria voidaan siis hyödyntää myös web-sovellusten kehityksessä, kunhan oikea jako asiakkaan ja palvelimen välillä on selvä, ja käytettävät tekniikat tukevat sitä. [38]

3.2 Web-sovellus

Web-sovellus on asiakas-palvelinmallia noudattava sovellus, jossa asiakaspuolen osa ohjelmistosta ajetaan verkkoselaimessa. Yleisiä web-sovelluksia ovat esimerkiksi sähköpostilaatikot, verkkokaupat ja verkkohuutokaupat. Dynaaminen verkkosivu pystyy taas muokkaamaan esitettämäänsä sisältöä skriptien eli lyhyiden koodinpätkien avulla. Dynaamisten verkkosivujen ja web-sovellusten välinen raja on epäselvä. Eniten sovellusten kaltaisia ovat yhden sivun sovellukset (single-page application), koska niissä eri sivuja ei tarvitse erottaa eri URL-osoitteilla. Esimerkiksi luvussa 4.1.5 kuvattu AngularJS on tälläisten web-sovellusten ohjelmistokehys. [28]

3.3 Käytetyt tekniikat

Luvussa kuvataan web-palvelun toteutuksessa käytetyt teknologiat ja ratkaisut. Käytettyjä teknologioita ovat JavaScript-ohjelmointikieli, JSON-tietorakenne ja NoSQL-tietokanta. Luvussa kuvataan lyhyesti myös relaatiotietokanta, joka auttaa ymmärtämään NoSQL-tietokannan tarvetta ja rakennetta. Luvussa selvennetään myös pilvilaskentaa ja pilvipalvelumalleja, koska toteutettu web-sovellus toteutettiin pilvipalveluksi.

3.3.1 JavaScript

JavaScript on korkean tason olio-ohjelmointikieli. JavaScript luotiin aluksi skriptauskieleksi tuottamaan elävyyttä web-sivuille. Myöhemmin JavaScript on laajentunut täysin yleiskäyttöiseksi ohjelmointikieleksi. JavaScript on web-kehittäjien eniten käyttämä teknologia ja maailman käytetyin ohjelmointikieli. [9] JavaScript on yksi ECMAScript-kielen toteutuksista. ECMAScript-kielen spesifikaation määrittelee Ecma International standardissa ECMA-262. ECMAScriptiä toteuttaa JavaScriptin lisäksi muun muassa Microsoftin JScript. [33]

JavaScript on tulkattava, dynaaminen skriptauskieli. Syntaksi on tarkoituksenmukaisesti tehty muistuttamaan C++:aa ja Javaa, jolloin uusien käsitteiden opettelun tarve on pienempi JavaScriptiin tutustuessa. Ehtolauseet, silmukkarakenteet ja muutamat muut toiminnallisuudet toimivat samoin, tai lähes samoin kuin edellä mainituissa kielissä. Syntaksiesimerkki nähdään listauksessa 3.1. JavaScriptiä käytetään yleisimmin osana verkkoselaimia. Selaimien JavaScript-toteutuksia ovat esimerkiksi Mozilla Foundationin SpiderMonkey, jota käytetään Firefox-selaimessa, Googlen V8, jota käytetään Chrome- ja Opera-selaimissa, ja Microsoftin ChackraCore, jota käytetään Edge-selaimessa. JavaScript-toteutuksia kutsutaan myös JavaScript-moottoreiksi. Toinen yleinen JavaScriptin käyttötarkoitus on toimia myös palvelinpuolen skriptauskielenä. Node.js, joka käyttää Googlen V8:aa, on tästä hyvä esimerkki.[1, 8]

```
1 // Tulostaa konsoliin "Hello World!"
  console.log("Hello World!");
3
  // Yksinkertainen rekursiivinen funktio
5 function kertoma(n) {
      if (n == 0) {
7         return 1;
      }
9
      return n * kertoma(n - 1);
11 }
```

Listaus 3.1 *JavaScript-kielen esimerkkejä, Hello World ja rekursiivinen funktio*

JavaScriptin kehitti alunperin Netscapen työntekijä Brendan Eich vuonna 1995. JavaScriptin aiempia nimiä ovat olleet Mocha ja LiveScript, mutta nimi vaihtui Sun Microsystemsin ja Netscapen lisensointisopimuksen jälkeen JavaScriptiksi. JavaScript ei itsessään liity Java-ohjelmointikieleen, vaan sen oli tarkoitus toimia Javan rinnalla skriptikielenä. Kielen standardointi alkoi vuonna 1996, jolloin julkaistiin ensimmäinen versio ECMA-262 standardista. [37]

3.3.2 JavaScript Object Notation

JavaScript Object Notation eli JSON on kevyt, yksinkertainen, ohjelmointikielestä riippumaton ja tekstipohjainen formaatti datan siirtämistä varten. Se määriteltiin alunperin ECMA-262 standardin kolmannessa versiossa vuonna 1999. JSON käyttää rakenteita, jotka ovat tuttuja useista ohjelmointikielistä, kuten C, C++, C#, Java, JavaScript, Perl ja Python. Näitä rakenteita on kaksi, joista ensimmäinen on nimi-arvo -pari, joka tunnetaan ohjelmointikielissä muun muassa nimillä objekti, struct ja dictionary. Toinen rakenne on järjestetty lista, joka tunnetaan muun muassa nimillä jono, vektori ja lista. [12]

Nykyään JSONin määrittelee Ecma Internationalin ylläpitämä ECMA-404 standardi. [32] Esimerkki JSON-formaattia noudattava objektista on nähtävissä listauksessa 3.2. JSON-objektin esitysmuoto on aina Unicode-merkistöä käyttävä teksti.

```
{
"id": 1,
"nimi": "Matti Meikäläinen",
"tiedot": {
    "sukulaiset": [2, 3, 4, 6, 8],
    "kulkuneuvot": [
        {"auto": true},
        {"polkupyörä": false}
        ]
    }
}
```

Listaus 3.2 *Esimerkki JSON-objektista*

JSON-objektilla on standardoitu rakenne. JSON-objekti on aaltosulkeiden välissä oleva merkkijonon, kaksoispisteen ja arvon yhdistelmä, joita voi olla useampia peräkkäin. Tällöin kyseiset yhdistelmät erotetaan toisistaan pilkulla. Objektin ei ole pakko sisältää edellä kuvattua yhdistelmää, vaan voi olla myös tyhjä. Yhdistelmässä kuvattu arvo voi JSON-objektissa olla toinen objekti, merkkijono, luku, lista tai jokin varatuista arvoista *true, false* tai *null*.

3.3.3 Relaatiotietokanta

Relaatiotietokannan esitteli IBM:n työntekijä E. F. Codd artikkelissa "A Relational Model of Data for Large Shared Data Banks" vuonna 1970. [31] Coddin esittelemässä relaatiomallissa käyttäjä ei pysty tekemään muutoksia, jotka rikkoisivat datan tai sen esitysmuodon. Relaatiomalli järjestää tietokannan tauluihin, eli relaatioihin. Relaatiot sisältävät rivejä ja sarakkeita. [21] Jokaisen rivin identifioi uniikki avainarvo. Jokainen rivi esittää kyseisen relaatiotyypin instanssia, esimerkiksi tiettyä henkilöä tai tiettyä tuotetta. Sarakkeiden avulla kuvataan kyseisten instanssien ominaisuuksia, kuten hintaa, osoitetta tai mallia. Jokaisella instanssilla on pääavain, edellä mainittu uniikki avainarvo. Taulukossa 3.2 on nähtävissä esimerkki yhdestä relaatiotietokannan taulusta. Esimerkin pääavaimena voisi toimia id-niminen ominaisuus.

Taulukko 3.2 *Esimerkki relaatiotietokannan taulusta*

id	etunimi	sukunimi	email
1	Antti	Virtanen	antti.v@mail.fi
2	Pentti	Lahti	pena.lahti@mail.fi
3	Ritva	Sjöblom	ritva@sjomail.fi
4	Jouko	Korhonen	jou@ko.net

Instansseilla voi olla myös muita avaimia, joita kutsutaan vaihtoehtoisiksi avaimiksi. Esimerkiksi henkilötunnus, jota ei käytetä relaatiossa pääavaimena, voi olla luonnollisempi tapa identifioida henkilö ja toimii silloin vaihtoehtoisena avaimena. Relaatiotietokantaan voidaan luoda suhteita eri relaatioiden välille käyttämällä pääavaimia tai vaihtoehtoisia avaimia.

Käytännössä kaikki relaatiotietokannat käyttävät SQL-kyselykieltä, jonka avulla tietokantaan voi tehdä lisäyksiä, hakuja ja muutoksia. Relaatiotietokannat toteuttavat yleensä ACID-ominaisuuksia, jotka takaavat tietokannan toiminnan olevan luotettavaa ja johdonmukaista. ACID-ominaisuudet ovat Atomicity eli atomisuus, Consistency eli johdonmukaisuus, Isolation eli eristettävyys ja Durability eli pysyvyys. [35]

3.3.4 NoSQL-tietokanta

NoSQL- eli Not Only SQL -tietokanta kuvaa tietokantaa, joka on perinteisestä relaatiotietokannasta poikkeava. NoSQL-tietokannat eivät yleisesti käytä SQL-kyselykieltä. NoSQL-tietokantojen hyödyllisin käyttötarkoitus on toiminta suurten datamäärien kanssa, kun data ei vaadi relaatiomallien käyttöä. Seuraavat kappaleet perustuvat lähteeseen [40].

NoSQL-tietokantojen toimintaa kuvaa CAP-teoreema. CAP-teoreemalla tarkoitetaan kolmea ominaisuutta, joista vain kaksi voidaan saavuttaa täysin ja samanaikaisesti. Kolmesta ominaisuudesta ensimmäinen on vahva johdonmukaisuus (*Consistency*). Tämä tarkoittaa sitä, että kaikki käyttäjät näkevät saman informaation. Toinen ominaisuus on korkea saatavuus (*Availability*), eli järjestelmä toimii myös virheiden sattuessa. Viimeinen ominaisuus on ositustoleranssi (*Partition tolerance*), eli tietokannan hajautus ei vaikuta toimintaan ongelmien sattuessa.

Ominaisuuksia voidaan verrata relaatiotietokantojen ACID-vaatimuksiin. CAP-teoreeman ominaisuudet eivät takaa samoja ominaisuuksia kuin ACID-ominaisuudet, mutta ovat samalla tavalla informatiivisia tietokannan suorituskykyä kuvailtaessa. NoSQL-tietokannat voidaan luokitella neljään eri tyyppiin, joista jokainen on optimoitu eri tarkoitusta varten: avain-arvomalli, dokumenttimalli, sarakemalli ja graafimalli.

Avain-arvomallissa säilötään avaimia ja niitä vastaavia arvoja yksinkertaisissa taulukoissa, joita kutsutaan myös hajautustauluiksi. Taulukko 3.3 kuvaa esimerkkiä yhdestä hajautustaulusta. Hakuja voidaan yleensä tehdä vain kokonaisten avaimien perusteella. Arvot voivat olla yksittäisiä merkkijonoja tai monimutkaisempia

listoja. Avain-arvomallia käyttävät tietokannat sopivat parhaiten erittäin nopeaa ja skaalautuvaa arvojen noutamista varten, esimerkiksi käyttäjäprofiilien hallintaan.

Taulukko 3.3 *Esimerkki Avain-arvo -taulukosta*

avain	**arvo**
1	Valmistaja: Opel Malli: Astra Väri: Vihreä Vuosi: 2015
2	Valmistaja: Opel Malli: Insignia Väri: Ruskea Vuosi: 2014

Dokumenttimallin tietokannat on suunniteltu hallitsemaan ja tallentamaan dokumentteja, jotka on koodattu jossain standardissa datansiirtoformaatissa, kuten XML tai JSON. Dokumentti voi olla rakenteeltaan monimutkaisempi kuin avain-arvomallin taulukossa ja sisältää satoja ominaisuuksia. Listaus 3.2 sisältää esimerkin JSON-muotoisesta dokumentista. Dokumenttien rakenne ei myöskään tarvitse olla keskenään samanlainen, vaan dokumenttien ominaisuudet voivat erota toisistaan. Tämän vuoksi dokumenttitietokanta sopii hyvin käsittelemään dataa, joka sisältää paljon puuttuvia arvoja tai on muodoltaan epäsäännöllistä. Dokumenttitietokannoissa kaikki avaimet ja arvot ovat täysin haettavia.

Sarakemallin tietokannat ovat joko avain-arvotyyppisiä tietokantoja, jotka sisältävät aikaleimatoiminnallisuuksia, tai Googlen Bigtable -mallin mukaisia, hakuindeksointiin optimoituja tietokantoja. Mahdollisuus aikaleimata tietokannan tiedostoja tekee sarake-mallin tietokannoista hyviä hajautetulle ja versiointia tarvitsevalle datalle, sekä sen prosessoinnille. Saraketietokannat on optimoitu suurille ja hajautetuille tiedostojärjestelmille, kuten esimerkiksi Googlen GFS:lle.

Graafimallin tietokannat korvaavat relaatioita sisältävät taulut rakenteellisia relaatioita sisältävillä graafeilla. Graafit esitetään solmujen ja niiden välisten suhteiden avulla. Graafin solmut ovat tietokannan objekteja. Graafitietokantojen pääpaino on ihmisystävällisessä informaation visuaalisessa esityksessä. Graafitietokantoja käytetään, kun relaatiot objektien välillä ovat kiinnostavampia, kuin itse objektit. Graafi-mallin tietokannat soveltuvat esimerkiksi sosiaalisten verkostojen hallintaan.

Relaationaalisen datan käsittelyä NoSQL-tietokannassa voidaan tehdä kolmen eri tekniikan avulla. Useiden kyselyjen käyttäminen kaiken datan noutamiseen on varteen otettava vaihtoehto, koska yksittäiset NoSQL-kyselyt ovat usein nopeampia suorittaa kuin perinteiset SQL-kyselyt. Jos kyselyjä täytyy tehdä suuri määrä, tätä

tekniikkaa on syytä välttää. Toinen tekniikka on tallentaa dokumenttiin tai objektiin toisen objektin ID:n lisäksi myös eniten tarvitut arvot. Esimerkiksi blogisivuston viestin sisältävä dokumentti voisi sisältää viestin lähettäneen käyttäjän ID:n lisäksi myös nimen, jotta nimeä tarvitse hakea pelkän viestin esitystä varten. Kolmas tapa on sisällyttää enemmän dataa objektiin tai dokumenttiin. Tällöin esimerkiksi sosiaaliseen mediaan lähetetyn päivityksen kommentit tallennettaisiin samaan dokumenttiin päivityksen kanssa. [17]

3.3.5 Pilvilaskenta ja pilvipalvelumallit

Pilvilaskenta kuvaa mallia, jossa palvelimet ja niiden ohjelmistot, esimerkiksi virtuaalikoneet, on pakattu käyttäjäystävällisempään ja kustannustehokkaampaan muotoon palveluksi, joka on käyttäjille helposti lähestyttävä. Internetin kehitys on mahdollistanut pilvilaskennan, kun käyttäjän ja palvelimen välinen viive on pienentynyt erittäin pieneksi suurien tiedonsiirtonopeuksien takia. Tällöin ei myöskään ole väliä, missä palvelin tarkalleen sijaitsee. Amazon oli ensimmäisiä yrityksiä, joka tarjosi pilvilaskentaa ostettavana palveluna, mutta Google ja Microsoft seurasivat pian perässä. Pilvilaskentaa hyödyntäviä, kuluttajille suunnattuja palveluita on nykyään useita, esimerkiksi tallennustilaa tarjoava Dropbox, elokuvien ja sarjojen striimauspalvelu Netflix ja valokuvien jakamiseen luotu palvelu Flickr. [43, s.1-3]

Internetin laaja saatavuus, suuret tiedonsiirtonopeudet ja virtuaalikoneiden konsepti mahdollistivat pilvilaskentapalveluiden synnyn. Nämä palvelut voidaan jakaa kolmeen eri pilvipalvelumalliin. [43, s.17-29] Ensimmäinen on **SaaS** eli *Software as a Service*, joka on pilvipalveluiden yleisin tyyppi. Se tarjoaa jonkin tietyn ohjelmiston web-palveluna. Yleinen esimerkki on organisaation sähköpostipalvelu. Muita yleisiä SaaS-palveluita ovat Microsoftin Office 365 ja Googlen Google Apps.

Toinen malli on **PaaS** eli *Platform as a Service*. Siinä pilvipalvelun tuottaja tarjoaa, lähinnä yrityksille, mahdollisuuden valita tarpeisiinsa sopivia ohjelmistoja palveluntarjoajan alustasta tai kehittää ohjelmistoja palveluntarjoajan alustalle. Yleensä palveluntarjoaja tarjoaa käyttäjille komponentteja, joista käyttäjä valitsee halutut ja tarpeelliset komponentit omaan ohjelmistokokonaisuuteensa. Esimerkkejä näistä ovat Googlen Google App Engine ja Salesforce.com.

Viimeinen malli on **IaaS** eli *Infrastructure as a Service*, joka tarjoaa infrastruktuurin pilvipalveluna, mahdollistaen täysin räätälöityjen palveluiden rakentamisen pilvipalveluun. Tarjotut palvelut voivat olla käytännössä samoja kuin PaaS-mallissa, mutta IaaS-mallissa käyttäjä on itse täysin vastuussa palvelun toiminnasta ja ylläpidosta. Tämä edellyttää IaaS-palveluja käyttävältä yritykseltä teknologista asiantunte-

musta. IaaS-mallia hyödynnetään myös usein hetkelliseen laskentaresurssien käyttämiseen, kun yritys ei itse halua investoida palvelimiin tai muuhun laitteistoon. Yleisimpiä IaaS-palveluntarjoajia ovat esimerkiksi Amazon, Xerox ja IBM.

4. PALVELUN TOTEUTUS

Luvussa kuvataan web-palvelun toteuttamiseen valitut ratkaisut, joihin kuuluvat MEAN-arkkitehtuurin neljä eri osaa ja Amazonin pilvipalveluja. MEAN-arkkitehtuurin osat ovat MongoDB NoSQL-tietokanta, palvelinpuolen ohjelmistokehys Express, asiakaspuolen ohjelmistokehys AngularJS ja Node.js-palvelinohjelmisto. Luvussa kuvataan myös web-palvelun tärkeimmät toiminnallisuudet, valittu pilvipalvelu ja projektin aikana toteutetun web-palvelun rajapinta. Luvussa 2.4 web-palvelulle esitettyihin vaatimuksiin vastataan luvussa esiintyvien kuvausten avulla.

4.1 Valitut ratkaisut

Ratkaisujen valinta perustui muutamaan argumenttiin. Ensimmäinen ja tärkein syy oli se, että kaikki tekniikat olivat Intopalon web-kehittäjille entuudestaan tuttuja vaihtoehtoja. Niitä oli käytetty useissa projekteissa, joten tukea niiden käyttöön oli saatavilla yrityksen sisältä. Valitut ratkaisut olivat myös globaalisti yleisiä ja niitä oli laajalti käytössä, joten kokemuksia kyseisten ratkaisujen käytöstä löytyi internetistä reilusti.

Valittujen ratkaisujen tuli olla helppokäyttöisiä. Teknologioiden tuli olla opittavissa lyhyellä itseopiskelulla ohjelmointitaitojen ollessa muuten hallussa. Intopalon vanhempien web-kehittäjien mukaan kyseiset ratkaisut täyttivät tämän vaatimuksen.

4.1.1 MEAN-arkkitehtuuri

Järjestelmän toteutusarkkitehtuuriksi valittiin MEAN-arkkitehtuuri, joka koostuu MongoDB NoSQL-tietokannasta, Express palvelinohjelmistokehyksestä, asiakaspuolen AngularJS ohjelmistokehyksestä ja Node.js-palvelinohjelmistosta. Se perustuu asiakas-palvelinarkkitehtuurille, jossa asiakkaan ja palvelimen välillä käytetään REST-rajapintaa. MEAN-arkkitehtuurin etuna on se, että sen kaikkia osia voidaan kehittää Javascript-ohjelmointikielellä. MEAN-arkkitehtuurin synty ja tämänhetkinen suosio onkin perua JavaScript-ohjelmointikielen yleistymisestä pelkästä skriptikielestä

yleiskäyttöiseksi ohjelmointikieleksi. JavaScript toteuttaa useampaa ohjelmointiparadigmaa, mutta MEAN-arkkitehtuuria käytettäessä toteutus täytyy olla asynkroninen ja tapahtumapohjainen.

MEAN-arkkitehtuuria toteuttava AngularJS-sovellus on siis asiakkaan puolelle lähetettävä itsenäinen sovellus, joka kommunikoi palvelimen kanssa. Kommunikointiin käytetään REST-rajapintaa. Arkkitehtuuri on tällä hetkellä paljon käytetty, mutta yhä kehityksen alla. Esimerkiksi AngularJS:n päivittyminen versioon 2 rikkoo taaksepäin yhteensopivuutta AngularJS 1.x versioiden kanssa. AngularJS voidaan korvata myös muilla vaihtoehdoilla, esimerkiksi Ember.js:llä, joka on hyvin samantyyppinen ohjelmistokehys kuin AngularJS. MEAN-arkkitehtuurin ympärillä oleva yhteisö on laaja, ja se perustuu avoimen lähdekoodin ratkaisuihin. [39]

4.1.2 MongoDB

MongoDB [14] on dokumenttimallinen NoSQL-tietokanta, joka kuvataan tarkemmin aliluvussa 3.3.4. MongoDB:ssä tiedot tallennetaan collection-nimisiin rakenteisiin. MongoDB:n collection eli kokoelma sisältää BSON-tyyppisiä (binary JSON) dokumentteja, jotka ovat JSON-tiedostoja binäärisessä esitysmuodossa. Dokumentin maksimikoko on 16 megatavua. MongoDB generoi jokaiselle dokumentille **_id**-nimisen ominaisuuden, joka identifioi jokaisen dokumentin. [15]

MongoDB:llä on useita etuja. MongoDB:tä voidaan käyttää JavaScript-ohjelmointikielellä, se on nopea pienillä tiedostomäärillä ja skaalautuu kooltaan jopa useaan petatavuun. Se oli myös vuoden 2015 lokakuussa suosituin dokumenttimallin NoSQL-tietokanta. BSON-tiedostomuoto mahdollistaa myös nopean tietokannan sisällön tuonnin ja viennin. [42]

4.1.3 Node.js

Node.js-palvelinohjelmisto [16] on arkkitehtuurin tärkein osa. Se on tehokas, asynkroninen ja tapahtumapohjainen palvelinohjelmisto. Node.js on kehitetty Googlen V8 JavaScript-moottoria hyödyntäen. REST rajapinnan kehittäminen Node.js:n avulla on tehokasta, varsinkin kun käytetään Express.js ohjelmistokehystä. Node.js on dynaamisen sisällön esittämiseen kilpailevia teknologioita tehokkaampi, ja sen JavaScript-toteutus on paljon nopeampi kuin perinteiset PHP-skriptikielellä toteutetut ratkaisut. [42]

4.1.4 Express.js

Express.js [11] on web-sovellusten palvelinohjelmistokehys, joka rakentuu Node.js:n toiminnallisuuksien päälle. Se tarjoaa kehittäjille yksinkertaisen tavan hallita sovelluksen reititystä ja HTTP-kutsuja, kuten GET ja POST. Express.js:n käyttö on hyödyllistä, koska sovelluksen kehitys on nopeampaa sen avulla. Express on myös aktiivisesti käytetty, ylläpidetty ja testattu, joten sen voi olettaa vakaaksi sovelluskehykseksi. [42, 10]

4.1.5 AngularJS

AngularJS [5] on web-sovellusten kehitysympäristö ja ohjelmistokehys. AngularJS rakentaa web-sovelluksen arkkitehtuurin käyttäen MVC-mallia. AngularJS:n avulla sovelluskehittäjät voivat kuvata, miten palvelimen data linkittyy tapahtumiin ja web-sovelluksen käyttöliittymäkomponentteihin, sekä miten tapahtumat ja käyttäjän toiminta muuttaa käyttölittymäkomponenttien tilaa. Tästä mallista syntyy yhden sivun sovellus (single-page application), joka ladataan kokonaisuudessaan asiakkaalle heti aluksi. Tämän jälkeen muu data saadaan JSON-objektina REST-rajapinnan kautta palvelimelta eli Node.js-sovellukselta. [39]

4.1.6 Käytetyt Amazonin pilvipalvelut

Amazon Web Services eli AWS oli vuonna 2014 maailman suurin pilvipalveluiden tarjoaja. Sen tarjoama laskentakapasiteetti oli viisi kertaa suurempi, kuin kaikkien muiden palveluntarjoajien yhteensä. Sillä on asiakkaita maailmanlaajuisesti yli 190 eri maassa. AWS tarjoaa SaaS-, PaaS- ja IaaS-palveluja. Esimerkiksi Dropbox pilvitallennustilan tarjoajan palvelu perustuu Amazonin S3-pilvitallennuspalvelun hyödyntämiseen käyttäjien tiedostoja hallitessa. [43, s.64-66]

Projektin aikana hyödynnettiin useita eri Amazon Web Services:in palveluja, kuten Amazon EC2:ta [2] palvelun tarjoamiseen, Amazon SES:ä [4] automaattisten sähköpostien lähettämiseen ja AWS Lambdaa [7] automaattisten varmuuskopioiden ottamiseen.

4.2 Toteutus

Aliluvussa kuvataan palvelun toiminnallisia yksityiskohtia. Aliluvussa esitellään lyhyesti minkälainen on rakennettu REST-rajapinta ja minkälainen on MongoDB-tietokannan kokoelmarakenne. Kuvauksessa 4.2.1 huomataan, kuinka yksinkertainen

ja intuitiivinen rakenne toteutetusta rajapinnasta syntyi.

4.2.1 Rajapintakuvaus

Web-palvelun rajapintojen suunnittelussa noudatettiin REST-arkkitehtuurimallia. REST perustuu HTTP-protokollan metodeihin, esimerkiksi GET, POST, PUT ja DELETE. Taulukossa 4.1 on listattu palvelun rajapinnat, joita voidaan kutsua. Taulukkoon on myös merkitty, millä HTTP-metodeilla mitäkin rajapintaa voidaan ohjelmallisesti kutsua.

Taulukko 4.1 *Palvelun REST-rajapintakuvaus*

HTTP metodit	**REST-rajapinta**
GET, DELETE	api/tokens
GET, POST	api/customers
GET, PUT, DELETE	api/customers/:id
GET, POST	api/dummies
GET, PUT, DELETE	api/dummies/:id
GET, POST	api/logs
GET, POST	api/records
GET, PUT	api/records/:id
GET, POST	api/trucks
GET, PUT, DELETE	api/trucks/:id
GET, POST	api/users
GET, PUT, DELETE	api/users/:id
POST	api/files/upload
GET	api/files/download/:name
GET, POST	api/files/metadata
GET, DELETE	api/files/metadata/:id

Rajapinnan kautta voidaan hallita tokeneja, joita hyödynnetään käyttäjien autentikointiin, asiakkaita, työkoneita, lokitietoja, tapahtumia, käyttäjiä sekä tiedostoja ja niiden metatietoja. Esimerkiksi haluttaessa kaikki työkoneet voidaan kutsua GET-metodilla rajapintaa */api/trucks* ja haluttaessa tietty työkone, kutsutaan silloin GET-metodilla rajapintaa */api/trucks/:id*, jossa :id on tietyn työkoneen ID. Kutsuttaessa DELETE-metodilla edellistä rajapintaa, saadaan poistettua kyseinen työkone. Token-rajapinnan (*api/tokens*) avulla käyttäjä voidaan autentikoida palveluun. Sitä kautta käyttäjä saa autorisoivan tokenin, jonka avulla voi käyttää muita rajapintoja. Files-rajapinnan (*api/files*) avulla palveluun voidaan ladata tiedostoja, tiedostojen metatietoja voidaan lisätä ja tiedostoja voidaan ladata palvelusta. Yleisesti POST on tietojen lisäämistä varten ja PUT päivittämistä varten.

4.2.2 Tietokantarakenne

Palvelun tietokantarakenne on yksinkertainen. Sen kokoelmia käsitellään luvussa 4.2.1 kuvatulla rajapinnalla Node.js serverin kautta. Taulukossa 4.2 on listattu palvelun MongoDB-tietokannan dokumenttikokoelmat. Kokoelmia on seitsemän ja ne sisältävät kaikki palveluun tallennetut tiedot. Palveluun tallennetut tiedostot tallennetaan tiedostojärjestelmään, mutta kaikkien tiedostojen metatiedot tallennetaan tietokantaan. Aliluvussa 4.3 on listattu toiminnallisuuksiin liittyvien dokumenttien esimerkkejä, joita kokoelmat sisältävät.

Taulukko 4.2 *MongoDB-tietokannan kokoelmat*

Kokoelma	**Sisältö**
customers	palvelun asiakkaiden tiedot
dummies	työkoneet, joista ei kerätä käyttötuntidataa
logs	etälaitteiden lähettämät viestit
metafiles	palveluun ladattujen tiedostojen metadata
records	palveluun kirjatut tapahtumat
trucks	työkoneet, joista kerättään käyttötuntidataa
users	palvelun käyttäjät: ylläpitäjät, huoltomiehet ja asiakkaat

4.3 Toiminnallinen kuvaus

Tässä aliluvussa kuvataan palvelun tärkeimmät ominaisuudet sekä niiden toiminnallinen kuvaus. Toteutuksien yksityiskohtia ei kuvata, vaan palvelun eri ominaisuuksien toimintaa palvelua käyttävän henkilön kannalta. Kuvausten avulla vastataan aliluvussa 2.4 esitettyihin web-palvelun vaatimuksiin.

4.3.1 Käyttäjien hallinta

Palvelun vaatimuksena oli käyttäjätilien hallinta. Käyttäjätyyppejä palveluun luotiin kolme, ylläpitäjä, huoltomies ja asiakas. Tämän vaatimuksen toteuttaa toteutettu käyttäjien hallinta.

Admin eli ylläpitäjä pystyy hallinnoimaan palveluun luotavia tunnuksia. Tunnuksia on kolmen tyyppisiä perustuen käyttäjän rooliin. Listauksessa 4.1 on esimerkki käyttäjädokumentista MongoDB-tietokannassa. Asiakasroolin käyttäjä liitetään johonkin järjestelmän asiakkaaseen, jolloin käyttäjä näkee ainoastaan kyseisen asiakkaan tiedot. Huoltomiesroolin käyttäjä näkee kaikkien asiakkaiden tiedot, mutta ei pysty muokkaamaan niitä. Huoltomieskäyttäjä pystyy lisäämään muistiinpanoja sekä korjaus- ja huoltomerkintöjä kaikille työkoneille. Admin-käyttäjät eli palvelun ylläpitäjät näkevät kaikki järjestelmään syötetyt tiedot sekä pystyvät muokkaamaan niitä. Tähän sisältyy koneiden, asiakkaiden ja käyttäjien luomista, muokkaamista ja poistamista.

```
{
    "_id" : ObjectId("57f76aa6e2db5904bc4bd734"),
    "name" : "Esko Esimerkki",
    "email" : "posti@asdf.com",
    "password" : "$2a$10$3BI21gVJ2uxKVzvZTdjusOIP...",
    "role" : "Customer",
    "customerId" : "577b9c8bf4078834052ab28d",
    "customerName" : "Pirkkalan ABC",
    "homePage" : "/customers/577b9c8bf4078834052ab28d",
    "visits" : {
        "current" : "2016-10-07T09:28:26.599Z",
        "previous" : "2016-10-07T09:28:17.939Z"
    }
}
```

Listaus 4.1 *Esimerkki käyttäjä-objektista*

Käyttäjätilin luonnissa vaadittavia tietoja ovat käyttäjätunnus, sähköposti ja salasana. Palvelun tämänhetkisessä versiossa ylläpitäjän täytyy luoda käyttäjälle salasana, eikä salasanaa pysty vaihtamaan. Tavoitteena on, että seuraavissa versioissa ylläpitäjän ei tarvitsisi asettaa salasanaa, vaan käyttäjä pystyy rekisteröitymään sähköpostilla toimitetun aktivointilinkin kautta ja asettamaan salasanansa. Kyseinen ominaisuus päätettiin jättää toteutettavaksi palvelun myöhemmissä versioissa ja on listattu palvelun jatkokehityskohteeksi. Asiakaskäyttäjää luotaessa liitetään aikaisemmin luotu asiakas eli yritys, jonka työkoneita huolletaan, käyttäjän tietoihin. Käyttäjän tietoihin tallennetaan siis asiakasnimi, asiakaan ID, sivu, jolle käyttäjä ohjataan hänen kirjautuessaan sisään sekä kaksi aikaleimaa, jotka kertovat milloin käyttäjä on viimeksi käyttänyt palvelua. Aikaleimatietojen avulla visualisoidaan tapahtumasivulla edellisen käynnin jälkeen lisätyt tapahtumat huomiovärillä. Näin käyttäjä erottaa selkeästi uudet tapahtumat.

4.3.2 Asiakkaiden hallinta

Palvelun vaatimuksena oli yrityksen asiakkaiden hallinnointia ja työkoneiden tehtyjen tapahtumien tarjoaminen asiakkaan nähtäväksi. Tämän vaatimuksen täyttämiseksi palveluun toteutettiin asiakkaiden hallinta. Asiakas-tyyppinen käyttäjä liitettiin asiakkaaseen, jolloin kyseinen käyttäjä näki tarvittavat tiedot.

Ylläpitäjät pystyvät hallinnoimaan palvelussa olevia asiakkaita samoin kuin käyttäjätileja, eli asiakkaiden lisääminen, muokkaaminen ja poistaminen on mahdollista. Asiakkaasta talletetaan vain nimi, osoite ja sähköpostiosoite, kuten on nähtävissä esimerkkidokumentissa listauksessa 4.2. Tietojen avulla asiakkaat yksilöidään, mutta tulevaisuudessa sähköpostiosoitteisiin voidaan lähettää automaattisia ilmoituksia esimerkiksi työkoneille tehdyistä huolloista. Asiakkaat ovat objekteja, jotka linkitetään mahdollisiin asiakaskäyttäjätileihin ja työkoneiden tietoihin.

```
{
    "_id" : ObjectId("577b9c8bf4078834052ab28d"),
    "name" : "Pirkkalan ABC",
    "address" : "Pirkkatie 3",
    "email" : "pirkat@abc.fi",
    "features" : {
        "hourLogging" : false
    }
}
```

Listaus 4.2 *Esimerkki asiakas-objektista*

Asiakaskäyttäjien käyttöliittymää voidaan muokata ominaisuuksien perusteella. Palvelun ensimmäisessä vaiheessa ominaisuuksia on yksi: käyttötuntiseuranta. Jos käyttötuntiseurantaa ei ole merkitty aktiiviseksi jollekin asiakkaalle, käyttöliittymässä ei näy käyttötuntidataan liittyviä tietoja.

4.3.3 Työkoneiden hallinta

Palvelun vaatimuksena oli työkoneiden hallinnointi ja niiden digitaalisena työkirjana toimiminen myös työkoneille, joissa ei olisi etälaitetta. Palvelussa huoltomiehen pitäisi pystyä merkitsemään koneelle huoltoja ja korjauksia, etälaitteen käyttötuntidata pitäisi päivittyä automaattisesti työkoneen tietoihin ja huoltojen pitäisi noudattaa työkoneen huolto-ohjelmaa. Nämä vaatimukset toteutettiin palvelun työkoneiden hallinnassa.

Web-palvelussa on kahdentyyppisiä työkoneita: käyttötuntien perusteella huolletta-

via ja tietyn aikavälein huollettavia. Näillä työkonetyypeillä on yhtäläisyyksiä koneiden perustietotoihin liittyen ja erovaisuuksia, jotka liittyvät huoltotietoihin. Ylläpitäjällä on oikeudet lisätä, muokata ja poistaa työkoneita palvelusta. Kaikista työkoneista tallennetaan työkoneen yksilöivä nimi, työkoneen omistava asiakas, huoltojen yhteydessä päivitettävät työkoneen käyttötunnit, ylläpitäjän mahdollisesti asettama viesti, konekohtaisia yhteystietoja, joihin automaattisia sähköposteja voidaan lähettää ja huoltomiesten mahdollisesti tekemät muistiinpanot. Lisäksi tallennetaan huolto-ohjeen ID. Huolto-ohje on palveluun tallennettu tiedosto, jonka metatiedot, esimerkiksi nimi ja sijainti, tallennetaan tietokantaan.

Käyttötuntien perusteella huollettavista työkoneista tallennetaan sekä etälaitteen että työkoneen huolto-ohjelman tietoja. Etälaitteesta tallennetaan etälaitteen yksilöivä ID sekä etälaitteen lähettämät käyttötunnit. Huolto-ohjelmaan liittyviä tietoja ovat seuraavan huollon osoittava luku ja lista huolloista, joissa jokaisesta kerrotaan huollon nimi ja huoltoväli. Listauksessa 4.3 on nähtävissä esimerkki käyttötuntien perusteella huollettavasta työkoneesta nimeltä "Trukki 1". Työkoneen omistaa "Pirkkalan ABC" ja siihen on kiinnitetty etälaite, jonka ID on "A001". Työkoneen huolto-ohjelma koostuu neljästä huollosta, kolmesta "A1" -huollosta ja yhdestä "A2" -huollosta, jotka tapahtuvat 300 käyttötunnin välein.

```
{
    "_id" : ObjectId("57836533441465410e4b14a4"),
    "name" : "Trukki 1",
    "customerId" : "577b9c8bf4078834052ab28d",
    "customerName" : "Pirkkalan ABC",
    "usageHours" : 324234,
    "boxId" : "A001",
    "boxHours" : 3407,
    "fixHours" : 235,
    "message" : "",
    "maintenance" : {
        "nextMaintenance" : 1,
        "program" : [
            {
                "type" : "A1",
                "interval" : 300
            },
            {
                "type" : "A1",
                "interval" : 300
            },
            {
                "type" : "A1",
                "interval" : 300
            },
            {
                "type" : "A2",
                "interval" : 300
            }
        ],
        "next" : 3,
        "fileId" : "578cdae356009d16cf68088f"
    },
    "contacts" : [
        {
            "email" : "korjaajamies@gmail.com"
        }
    ],
    "emailSent" : false,
    "memos" : [
        {
            "timestamp" : "2016-09-27T06:28:06.105Z",
            "message" : "Puuttuu rengas."
        }
    ]
}
```

Listaus 4.3 *Esimerkki käyttötuntien perusteella huollettavasta työkone-objektista*

Työkoneisiin, jotka huolletaan vain tietyn aikavälin jälkeen, ei kiinnitetä etälaitetta. Kyseisten koneiden huoltotietoihin sisältyy edellisen huollon päivämäärä ja aika, huoltoväli kuukausina sekä seuraavan huollon päivämäärä.

4.3.4 Tapahtuman lisääminen

Palvelun vaatimuksena oli huoltojen ja korjauksien merkitseminen työkoneille. Tapahtumat-ominaisuus palvelussa mahdollisti tämän vaatimuksen toteuttamisen. Myöhemmin asiakkaalta tuli esiin tarve hinnan asettamisesta tapahtumille, joka myös toteutettiin.

Työkoneille voidaan tehdä huoltoja ja korjauksia. Korjaukset ovat huolto-ohjelmaan kuulumattomia toimintoja, joita suoritetaan esimerkiksi jonkin työkoneen osan hajotessa. Korjaukset eivät siis vaikuta huolto-ohjelman kulkuun samalla tavalla kuin huollot. Kun työkoneelle lisätään huolto tai korjaus, tapahtuu kaksi asiaa: tietokantaan lisätään tapahtuma-objekti ja päivitetään työkoneen tietoja. Tapahtumasta on nähtävissä esimerkki listauksessa 4.4.

```
{
    "_id" : ObjectId("579efc62ff9e2e08d94720d9"),
    "machineId" : "578f2c7ca7e59205b46b7d71",
    "machineName" : "Kone 1",
    "customerId" : "577b9c8bf4078834052ab28d",
    "date" : "2016-08-01T07:38:10.780Z",
    "usageHours" : 4500,
    "actionType" : "Maintenance",
    "notes" : "Hilavitkutin on kerännyt karstaa melko paljon",
    "price" : 234
}
```

Listaus 4.4 *Esimerkki tapahtuma-objektista*

Tapahtuman tietoihin kuuluvat työkoneen nimi ja ID, jolle tapahtuma lisätään, sekä asiakkaan ID, jolle työkone kuuluu. Työkoneen nimi on lisätty objektiin vain, jotta nimeä ei tarvitsisi erikseen hakea ID-tiedon perusteella. Tämä vähentää tietokantahakujen määrää. Tapahtuman tietoihin kuuluvat myös päivämäärä- ja aikainformaatio, jolloin tapahtuma on lisätty, työkoneen senhetkiset käyttötunnit, tapahtuman tyyppi eli huolto tai korjaus, tapahtuman kuvaus, jonka tapahtuman lisääjä kirjoittaa kuvaamaan suoritettua toimenpidettä sekä hinta, jonka vain ylläpitäjä pystyy lisäämään tapahtumalle. Tapahtuman lisäämisen jälkeen ylläpitäjä, jolla on pääsy yrityksen laskutusjärjestelmään, hinnoittelee tapahtuman. Tällöin asiakkaat näkevät kunkin tapahtuman hinnan ja tarkemman kuvauksen tehdystä työstä. Web-

palvelua ei integroitu laskutusjärjestelmään, vaikka kyseinen kehitysmahdollisuus tunnistettiin projektin alkuvaiheessa. Tätä ei kuitenkaan koettu vielä tarpeelliseksi ominaisuudeksi.

Työkoneen tietoihin päivitetään samalla sen hetkiset käyttötunnit koneen omasta käyttötuntimittarista. Mittari on usein analoginen, kumulatiivisesti tunteja laskeva mittari. Mittarilta saatuihin arvoihin voidaan myös verrata etälaitteelta saatuja tuntimääriä, joiden pitäisi kasvaa suunnilleen yhtä nopeasti. Huollon yhteydessä myös nollataan huoltoon liittyvä käyttötuntilaskuri ja huolto-ohjelman tiedoissa siirretään seuraavan huollon ilmaiseva osoitin osoittamaan oikeaa huoltoa.

4.3.5 Käyttötuntiperusteinen huolto-ohjelma

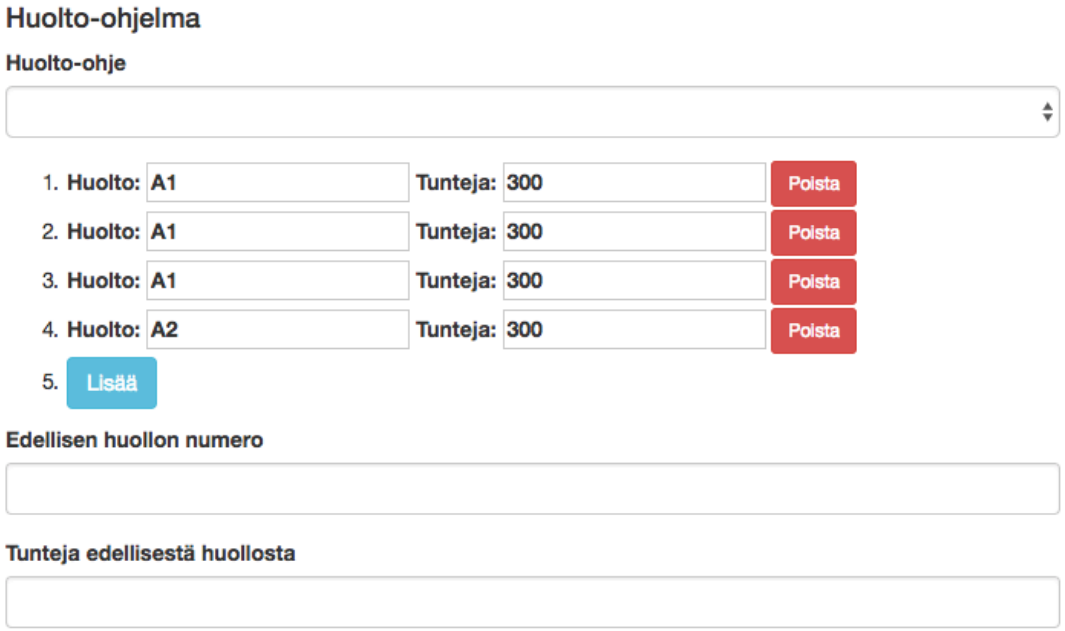
Palvelun vaatimuksena oli työkoneille suunniteltujen huolto-ohjelmien seuraaminen ja noudattaminen. Tämän vaatimuksen täytti palveluun toteutettu käyttötuntiperusteinen huolto-ohjelma

Jokaisen käyttötuntiperusteisesti huollettavan työkoneen tietoihin kirjataan työkonekohtainen huolto-ohjelma, jonka mukaan huollot suoritetaan. Huolto-ohjelmat koostuvat yleensä pienemmistä ja isommista huolloista, jotka on nimetty esimerkiksi tietyssä työkonetyypissä A1- ja A2-huolloiksi. Usein ohjelman mukaan suoritetaan muutama pieni huolto ennen isompaa huoltoa, tietyllä intervallilla. Yleinen huolto-ohjelma on esimerkiksi kolmensadan tunnin välein suoritettavat kolme pienempää A1-huoltoa, jonka jälkeen suoritetaan yksi laajempi A2-huolto. Kun työkone luodaan palveluun, samalla kirjataan myös koneen senhetkinen huoltotilanne. Käyttäjää pyydetäänkin tätä tarkoitusta varten syöttämään huolto-ohjelman lisäksi edellinen huolto ja kuluneet käyttötunnit edellisestä huollosta.

Esimerkki huolto-ohjelman määrittämisestä työkoneelle on nähtävillä kuvassa 4.1, jossa työkoneelle ollaan asettamassa edellä kuvattua huolto-ohjelmaa. Huolto-ohjelman ja koneen senhetkisen tilanteen lisäksi työkoneelle valitaan oikea huolto-ohje, joka on jo ladattu palveluun.

4.3.6 Huolto-ohjeiden hallinta

Palvelun vaatimuksena oli huolto-ohjeiden tallentaminen työkoneen tietoihin, jotta ne olisivat saatavilla palvelusta huoltoa suoritettaessa. Huolto-ohjeiden hallinta toteuttaa kyseisen vaatimuksen.



Kuva 4.1 *Huolto-ohjelman määrittäminen käyttötuntiperusteisesti huollettavalle työkoneelle web-palvelussa.*

Web-palveluun pystyy tallentamaan huolto-ohjeita työkoneiden huoltamisen helpottamiseksi. Työkoneiden huolto-ohjeet on laadittu eri konetyyppien mukaan, eli jokaisella koneella ei ole yksilöityä huolto-ohjetta. Konetyyppien määrä on myös pienehkö, jolloin niiden hallitseminen ja erottaminen on melko yksinkertaista. Palveluun siis ladataan huolto-ohjeita, joista perustettavalle koneelle valitaan oikea. Ohjeet ovat palvelimen tiedostojärjestelmässä talletettavia tiedostoja, joita ei ole rajoitettu tiedostotyypin suhteen. Web-palvelu ei esitä ohjetiedostoja, vaan ne täytyy ladata web-palvelusta luettaviksi. Tiedoston tallentamisen yhteydessä luodaan myös tietokantaan metatietoja sisältävä objekti, joka sisältää tiedoston nimen ja polun, johon tiedosto on tallennettu, sekä aikaleiman tiedoston tallentamisajankohdasta. Listauksessa 4.5 on esimerkki tietokantaan tallennettavasta metatieto-objektista.

```
{
    "_id" : ObjectId("57ad5a44c994ab0303ab42fe"),
    "name" : "Huolto_ohje.pdf",
    "path" : "./uploads/Huolto_ohje.pdf",
    "date" : ISODate("2016-08-12T05:10:28.746Z")
}
```

Listaus 4.5 *Esimerkki metatieto-objektista*

Web-palvelussa esitetään saatavilla olevat huolto-ohjeet metatietojen avulla. Meta-

tiedoissa olevan tiedostonimen avulla ylläpitäjä valitsee oikean tiedoston työkoneelle. Tiedostopolun avulla tiedosto löydetään ja voidaan ladata. Ylläpitäjä pystyy myös poistamaan tiedostoja web-palvelun kautta. Tällöin poistetaan molemmat, huolto-ohjetiedosto ja vastaava metatieto-objekti.

4.3.7 Etälaitteiden käyttötuntidata

Palvelun vaatimusten mukaan palvelun tulisi automaattisesti päivittää etälaitteelta tulevia käyttötunteja työkoneen tietoihin. Käyttötuntitiedot ovat vain etälaitteilta tulevia tietoja. Käyttäjät eivät pysty itse lisäämään koneille huoltoihin liittyviä käyttötunteja. Huoltomiehet ja ylläpitäjät voivat suorittaa työkoneelle huollon, jolloin palveluun syötetään työkoneen omasta käyttötuntimittarista kumulatiiviset, koko koneen elinaikana kertyneet käyttötunnit. Huollon tekeminen myös nollaa huolto-ohjelmaan liittyvän käyttötuntilaskurin. Etälaitteelta saapuvat käyttötunnit kasvattavat tätä lukua. Työkoneen tiedoissa säilytetään etälaitteelta viimeiseksi saapunutta arvoa ja tuntimäärää edellisestä huollosta.

Etälaite lähettää palveluun oman ID:nsä ja oman käyttötuntilaskurinsa arvon. ID:n avulla web-palvelussa etsitään työkone, jonka tiedoista löytyy vastaava etälaitteen ID. Web-palvelussa etälaitteilta saapuneet tiedot säilytetään lokiobjekteissa. Työkoneen ID ja aikaleima lisätään lokiobjektiin, joka tallennetaan tietokantaan. Esimerkki tallennettavasta objektista löytyy listauksesta 4.6. Aikaleima voitaisiin kerätä myös hetkestä, jolloin viesti lähtee etälaitteelta, mutta koska käyttötuntien päivittyminen ei ole kovin aikakriittistä, tehdään se vasta web-palvelussa. Tällöin kaikkien etälaitteiden tiedot saavat aikaleiman saman kellon mukaan. Työkoneiden ID:itä ei talleteta etälaitteisiin, jotta laitteen siirtäminen koneesta toiseen olisi mahdollisimman helppoa. Tällöin etälaitteen siirto onnistuu vain koneiden tietoja muokkaamalla web-palvelussa.

```
{
    "_id" : ObjectId("57863f53163cd71241489c26"),
    "machineId" : "5786174284d1df0bdf11424e",
    "boxId" : "A0001",
    "date" : ISODate("2016-07-13T13:17:07.460Z"),
    "boxHours" : 69
}
```

Listaus 4.6 *Esimerkki loki-objektista*

Kun etälaitteen lähettämä käyttötuntilukema on saapunut palveluun ja on löydetty työkone, jolle kyseinen etälaite on merkitty, verrataan saatua arvoa koneen tiedoissa

olevaan etälaitteen edelliseen lukemaan. Jos luku on sama tai suurempi, lasketaan vanhan ja uuden arvon erotus, jonka jälkeen tuntimäärää edellisestä huollosta kasvatetaan tällä luvulla ja tallennetaan se koneen tietoihin. Vanha etälaitteen käyttötuntilukema korvataan uudella.

Tämä logiikka mahdollistaa sen, että vaikka kaikki viestit eivät tule perille, käyttötuntilaskenta toimii silti. Jos uusi käyttötuntilukema on pienempi kuin edellinen, oletetaan, että etälaite on vaihdettu tai sen käyttötuntilaskuri on nollattu. Tällöin koneen tietoihin päivitetään vain etälaitteen käyttötuntilukema, eikä huoltoihin liittyvää käyttötuntilaskuria kasvateta. Kyseisessä tilanteessa "hukataan" vain ne tunnit, jotka vanha laite on jättänyt lähettämättä web-palveluun. Oikein toimiva laskenta alkaa heti ensimmäisen uuden laitteen viestin saapuessa web-palveluun. Haastavampi tilanne, johon ei ole toteutettua ratkaisua, on tilanne, jossa työkoneen etälaite vaihdetaan, mutta uuden etälaitteen käyttötuntilaskuri on paljon suuremmassa lukemassa, kuin edellisen laitteen. Tällöin käyttötuntien määrä hyppää näiden erotuksen verran ylöspäin. Ongelma on kuitenkin hetkellinen, sillä se ratkeaa tekemällä työkoneelle huolto, joka nollaa käyttötuntilaskurin.

4.4 Käyttöönotto

Vaadittujen ominaisuuksien valmistuttua asiakkaan yritys otti web-palvelun sisäiseen testikäyttöön. Testikäytön tarkoituksena on kerätä kokemuksia palvelun varsinaisilta käyttäjiltä, eli huoltomiehen roolissa toimivilta henkilöiltä. Palvelua ei lanseerata huoltoyrityksen asiakkaille ennen, kuin yrityksen työntekijät osaavat käyttää palvelua luontevasti, ja palvelu on havaittu luotettavaksi.

Web-palvelua testattiin ennen testijakson alkamista vain kehityksen ohessa. Asiakas testasi palvelun keskeneräisiä versioita muutaman kerran. Testijakson aikana oli tarkoitus myös löytää palvelun toimintaan vaikuttavat mahdolliset virheet, joita ei oltu havaittu kehityksen aikana. Testijakson aikana oli myös tarkoitus testata kehitettyjen etälaitteiden toimivuutta.

5. ETÄLAITE

Tässä luvussa kuvataan, miten käyttötunteja keräävä etälaite valittiin ja toteutettiin. Aluksi etälaitteelle asetettiin laitevaatimukset. Näiden vaatimusten perusteella tunnistettiin mahdolliset laitekategoriat, joista etälaite valittiin. Laitekategorioista löydettiin useita erilaisia vaihtoehtoja etälaitteeksi. Tämän jälkeen kuvataan miten mahdollisia laitevaihtoehtoja rajattiin, ja miten rajatusta joukosta valittiin kehitettävä etälaite. Luvussa kuvataan myös valitulle etälaitteelle kehitetyn ohjelmiston toimintalogiikkaa sekä etälaitteen testausta ja käyttöönottoa.

Etälaitteen korkean tason toimintalogiikka oli yksinkertainen. Laite laski työkoneen käynnissäoloaikaa aina virtaa saadessaan. Tämä kuvasi työkoneen käyttötunteja. Käyttötunnit talletettiin laitteen pysyväismuistiin ja lähetettiin tietyin väliajoin pilveen eli web-palveluun.

5.1 Laitevaatimukset

Etälaitteen minimiominaisuuksiksi listattiin ohjelmoitavuus, 2G-yhteys ja kirjoitettava pysyväismuisti. Ohjelmoitavuus asetettiin vaatimukseksi, jotta laite toimii valitun palvelun kanssa riippumatta siitä, onko palvelu itse tehty tai valmis ratkaisu. Tällöin myös viestien lähettämisen intervallin pystyy säätämään sopivaksi. Viestienvälitystavaksi valittiin GSM- eli 2G-datayhteys, koska työkoneet eivät yleisesti ottaen ole minkään muun tietoliikenneverkon peittoalueella. Työkoneet toimivat etäisissä paikoissa, missä matkapuhelinverkon kuuluvuus on epävarmaa. Tämän takia laitteesta on löydyttävä myös kirjoitettavaa pysyväismuistia. Laite pystyy tällöin tallentamaan pysyväismuistiin kuluneet käyttötunnit siihen asti, kunnes laite on taas verkon katvealueella. Näin laitteen luotettavuus kasvaa huomattavasti.

Erityistä huomiota vaativaksi näkökulmaksi tunnistettiin tulevaisuuden jatkokehitysmahdollisuudet eli laitteen muokkaaminen ja päivittäminen. Pitääkö kyseiseen skenaarioon valmistautua, vai riittääkö pelkkien käyttötuntien kerääminen myös tulevaisuudessa? Samalla huomioitiin myös etälaitetta koskevia avainkysymyksiä, jotka vaikuttaisivat laitevalintaan. Laitteen kestävyysvaatimukset eli koteloinnin toteutus, esteettiset vaatimukset, laitteen hinta ja minkälainen olisi käytettävä virtalähde.

5.2 Laitekategoriat

Selvitysvaiheen aikana tunnistettiin neljä eri kategoriaa, joihin etälaitelaitevaihtoehdot voitiin sijoittaa. Laitekategoriat on esitetty taulukossa 5.1. Ensimmäiset kaksi kategoriaa koostuivat valmiista kehitysalustasta. Valmiissa kehitysalustoissa, kuten Raspberry Pi:ssä [20] ja Arduinossa [6], voidaan käyttää joko valmista tai itse suunniteltua ja valmistettua koteloa. Valmiin kehitysalustan käyttämisessä hyötyjä tuo se, että itse alusta on kohtuuhintainen, niitä on hyvin saatavilla ja niistä on paljon kokemuksia. Haittoina tiedetään, ettei niiden kestävyydestä pitkällä aikajänteellä ole takeita, jolloin laitteen täytyykin olla helposti korvattavissa uudella. Yksi vaihtoehto laitteen koteloinnille oli itse suunniteltu muovi- tai metallikotelo. Itse suunniteltu kotelo mahdollistaisi laitteen brändäämisen juuri halutunlaiseksi, mutta muottien valmistaminen tuotantoa varten sisältää niin suuret kustannukset, ettei vaihtoehto ole kilpailukykyinen, ennen kuin tuotantomäärät kasvavat kymmeniin tuhansiin. Molemmissa laitekategorioissa laitteet täytyy koota itse. Asiakas hylkäsi ajatuksen itse tehdystä kotelosta melkein heti, koska etälaitteiden kappalemäärät olisivat enintään joitain satoja.

Taulukko 5.1 *Etälaitekategoriat*

Etälaite tyyppi	Kuvaus
Valmiiksi koteloitu kaupallinen kehitysalusta	Valmis kehitysalusta yhdistettynä valmiiseen koteloon. Ei sisällä virtalähdettä.
Kaupallinen kehitysalusta ilman kotelointia	Valmis kehitysalusta yhdistettynä räätälöityyn koteloon. Ei sisällä virtalähdettä.
Räätälöity laite	Tarkoitukseen räätälöity, uniikki laite. Sisältää koteloinnin ja virtalähteen.
Kaupallinen IoT-laite	Valmis, kaupallinen laite. Sisältää koteloinnin ja virtalähteen.

Kaksi muuta kategoriaa, jotka eivät perustu valmiisiin kehitysalustoihin, olivat räätälöity laite ja kaupallinen IoT-laite. Räätälöity laite tarkoittaa laitteita valmistavan yrityksen juuri tähän käyttötarkoitukseen suunnittelemaa ja valmistamaa laitetta. Räätälöidyn laitteen toiminnalla olisi laitteen valmistajan antama luotettavuustakuu. Tässä kategoriassa laitteen kustannuksiin tulisi mukaan usein laitteen kehitystyö, joka vaihtelisi laitevalmistajan ja työmäärän mukaan. Useiden yritysten tarjoamia vaihtoehtoja tutkittiin räätälöityjen laitteiden osalta. Huomattiin, että kehityskustannukset nostavat laitteen kappalehintaa huomattavasti, kun valmistettavat laite-erät ovat pieniä.

Valmiita IoT-laitteita löytyi markkinoilta useisiin erilaisiin tarkoituksiin, mutta ei

juuri tämän projektin tarkoitukseen. Mahdollisista vaihtoehdoista etsittiin tähän käyttötarkoitukseen sopivaa laitetta. Valmiista IoT-laitteista huomattiin, että niiden hintaskaala oli suuri, niissä oli ylimääräisiä ominaisuuksia tämän projektin käyttötarkoitusta varten ja usein laitteen myötä olisi joutunut käyttämään myös valmistajan omia palveluita. Esimerkki kaupallisesta IoT-laitteesta on Thingsee One Lite [24].

5.3 Laitevaihtoehdot

Asiakkaan mielikuva ja tavoite laitteen hinnasta liikkui selvästi alle kahdensadan euron hintaluokassa, mikä tarkoitti käytännössä sitä, että valmiiden IoT-laitteiden joukosta ei löytynyt tähän käyttötarkoitukseen sopivia vaihtoehtoja. Samoin suuri osa räätälöidyistä laitteista suljettiin pois suurten kehityskustannusten takia, vaikka itse laitteet olisivatkin olleet hinnaltaan sopivia. Räätälöityjä laitteita valmistavan yrityksen valikoimista löytyi kuitenkin yksi potentiaalinen vaihtoehto, joka otettiin mukaan mahdollisiin vaihtoehtoihin erilaisten kehitysalustavaihtoehtojen kanssa. Kyseinen laite oli aikaisemmin suunniteltu lähes vastaavaan tarkoitukseen, joten kehityskustannukset eivät näkyisi merkittävästi laitteiden hinnassa. Kehitysalustavaihtoehdoista löytyi laitteita, joissa pyöri Linux-käyttöjärjestelmä, esimerkiksi Raspberry Pi. Raspberry Pi ei sisältänyt 2G-modeemia. Kehitysalustavaihtoehtoihin kuului myös laitteita, joissa oli jokin matalamman tason ohjelmisto ja modeemi. Yhdessä alustassa oli modeemin lisäksi jopa GPS-vastaanotin.

Potentiaalisimmat kehitysalustavaihtoehdot olivat Raspberry Pi [20], Arduino [6], u-blox C027 [25] ja Particle Electron [19]. Yhteenveto laitteista on esitetty taulukossa 5.2. Raspberry Pi oli joukon ainoa, joka käyttää Linux-käyttöjärjestelmää. Muissa laitteissa oli matalamman tason ohjelmisto. Raspberry Pissä ja Arduinossa ei ollut integroitua GSM-modeemia, joten niihin se täytyisi lisätä yhteensopivien moduulien avulla. Particle Electron ja u-blox C027 sisälsivät integroidut GSM-modeemit. U-bloxin C027, oli ainut laite jossa oli integroitu GPS-vastaanotin, jonka avulla saataisiin laitteen sijaintitieto. Kyseinen laite perustui ARM-pohjaiseen mbed-alustaan [26]. Mikään laite ei sisältänyt työkoneeseen suoraan sopivaa virtalähdettä, vaan ne tulisi valita tai toteuttaa erikseen.

Näistä optimaalisimmaksi vaihtoehdoksi valittiin Particle Electron. Laitteessa oli integroituna modeemi, joten siihen ei tarvitsisi kiinnittää erillisiä komponentteja. Integroidun modeemin omaava laite olisi työkoneisiin kiinnitettäessä todennäköisesti kestävämpi vaihtoehto kuin Raspberry Pi tai Arduino, jotka vaatisivat erillisen modeemin. Particle Electronissa ei ollut myöskään ylimääräisiä integroituja antureita. Ne olisivat todennäköisimmin nostaneet laitteen hintaa. Esimerkiksi u-bloxin

Taulukko 5.2 *Oleellisimmat kehitysalustavaihtoehdot, niiden lyhyet ominaisuuskuvaukset ja hinta-arviot modeemin kanssa*

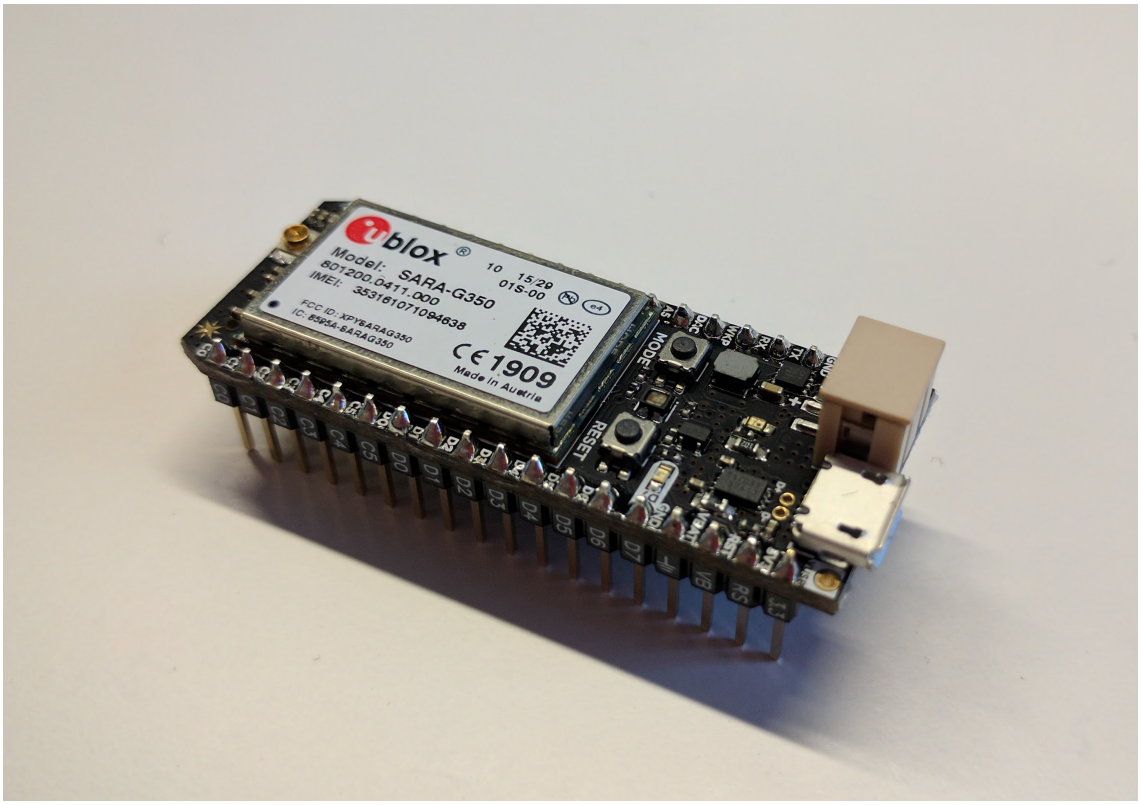
Etälaite	**Kuvaus**	**Hinta-arvio euroissa**
Raspberry Pi	Linux-käyttöjärjestelmä Ei antureita Ei modeemia	60-80 (modeemilla)
Arduino	Esim. C/C++:lla kirjoitettu ajettava ohjelma Ei antureita Ei modeemia	60-80 (modeemilla)
u-blox C027	mbed-ohjelmistoalusta GPS 2G-modeemi	99
Particle Electron	C/C++-kielen ajettava ohjelma Ei antureita 2G-modeemi	45

C027 oli yli kaksi kertaa kalliimpi kuin Particle Electron. Particle Electronissa oli myös useita liitäntöjä, joihin oli mahdollista liittää muita antureita tai moduuleita tarpeen vaatiessa. Laite oli myös vaihtoehdoista edullisin.

5.3.1 Laitevalinta

Selvitysvaiheen lopuksi asiakkaalle esiteltiin kaksi etälaitevaihtoehtoa. Kehitysalustoista varteenotettavin vaihtoehto oli Particlen Electron-niminen kehitysalusta, joka on samantyyppinen kuin Arduino, mutta siihen on integroitu 2G-modeemi. Laite on esitetty kuvassa 5.1. Laitetta on mahdollista ohjelmoida C- tai C++-ohjelmointikielellä setup- ja loop-funktioiden avulla. Laite suorittaa käynnistyessään setup-funktion ja tämän jälkeen loop-funktiota niin kauan, kuin saa virtaa. Laite ei sisältänyt mitään ylimääräistä projektin käyttötarkoitusta varten ja täytti aluksi asetetut minimivaatimukset. Laitteessa on myös liitäntämahdollisuudet kiinnittää erilaisia antureita. Tuote oli markkinoilla melko uusi, eikä laitteesta ollut paljon käyttökokemuksia tai takeita siitä, että se tulisi toimimaan tässä käyttötarkoituksessa.

Particle Electron täytyi liittää Particlen pilvipalveluun Particle Cloudiin [18]. Pilvipalvelu tarjosi ominaisuuksia kyseistä pilvipalvelua tukevien etälaitteiden hallintaan. Näitä ominaisuuksia olivat esimerkiksi laitekohtainen monitorointi, laitteiden ohjelmiston etäpäivitykset ja integroinnit muihin verkkopalveluihin. Particle Electron ei voi suoraan lähettää salattuja viestejä muihin verkkopalveluihin, mutta pilvipalvelun kautta se onnistuu. Tällöin on käytettävä Particle Cloud -pilvipalvelun Webhooks-nimistä ominaisuutta [29], joka mahdollistaa pilvipalveluun saapuvan



Kuva 5.1 *Particle Electron*

viestin lähetettäväksi edelleen Webhookissa määritettyyn verkko-osoitteeseen salattuna. Webhooks-ominaisuuden avulla on myös mahdollista lähettää viestejä Particle Cloudin kautta laitteisiin ja ohjata niiden toimintaa.

Toinen varteenotettava vaihtoehto oli Ionsignin valmistama, lähes vastaavaan tarkoitukseen valmistettu räätälöity laite. Laite on esitetty kuvassa 5.2. Alunperin laite on valmistettu trukkien törmäysten havaitsemiseen ja seurantaan. Laitteessa on 2G-modeemi ja sisältää mahdollisuuden kiinnittää muitakin antureita jatkokehitysmahdollisuuksia ajatellen. Laite sisältää tarpeelliset elektroniikat virransyöttöä varten työkoneiden vaihtelevista jännitteistä ja on kyseistä käyttötarkoitusta varten koteloitu. Laitteen heikoin puoli on sen ohjelmoitavuus. Ohjelmointi pitää tehdä tekstiviestien välityksellä, eikä laitteeseen pysty ohjelmoimaan kuin IP-osoite, johon viestit lähetetään ja viestien lähettämisintervallin. Laite on myös Particle Electroniin verrattaessa kalliimpi, hinnaltaan noin 150 euroa.

IonSignin etälaite ei pakottanut käyttämään erillisiä pilvipalveluita, vaan pystyi lähettämään viestit suoraan haluttuun IP-osoitteeseen. Jotta viestit olisi pystytty lähettämään salattuina, olisi laitteen modeemi pitänyt päivittää 2G-modeemista 3G-modeemiksi. Ionsignin mukaan modeemin vaihto ei olisi nostanut kehityskustannuk-

Kuva 5.2 *Ionsignin etälaitevaihtoehto.*

sia tai laitteen hintaa merkittävästi.

Näiden tietojen perusteella asiakas valitsi etälaitteen, josta kehitettäisiin työkoneiden käyttötunteja mittaava etälaite. Asiakas valitsi laitteeksi Particle Electronin. Asiakkaan tärkeimpänä kriteerinä laitteelle oli kappalehinta. Jatkokehitysmahdollisuuksia asiakas ei tässä laitevalinnassa priorisoinut. Asiakas myös hyväksyi sen, että laite täytyisi koteloida itse ja laitteelle täytyi etsiä erillinen virtalähde.

5.4 Particle Electron

Particle Electron osoittautui toimivaksi ensimmäisissä testeissä. Haasteita ja kysymyksiä kyseisen laitteen käytössä on laitteen kotelointityö, joka täytyy tehdä itse. Lisäksi kysymyksenä oli laitteen elinikä, josta ei voida antaa arvioita ennen pidempiä käyttökokemuksia. Valmis laite koostuu Particle Electronista, jännitemuuntimesta (esitetty kuvassa 5.3), joka muunsi työkoneiden 12, 24 tai 48 voltin jännitteet Particle Electronille sopivaksi 5 voltin jännitteeksi, ja kotelosta.

Kuva 5.3 *Etälaitteessa käytetty jännitemuunnin*

5.4.1 Laitteen toimintalogiikka

Particle Electronin ohjelmointi tapahtuu Particlen omassa web-palvelussa, jossa kirjoitetaan ohjelman toimintalogiikka C-ohjelmointikielellä. Web-palvelussa voidaan tämän jälkeen kääntää ja ladata kirjoitetun ohjelman binääritiedosto. Komentorivityökalun avulla ohjelmisto asennetaan laitteeseen USB-yhteyden yli.

Electronia voidaan käyttää kolmessa eri tilassa: automaattisessa, puoliautomaattisessa tai manuaalisessa [13]. Automaattinen tila alkaa automaattisesti käynnistyksen yhteydessä suorittamaan käyttäjän kirjoittamaa setup-funktiota, jossa se yhdistää laitteen matkapuhelinverkkoon. Tämän jälkeen suoritus siirtyy loop-funktioon, jota se toistaa niin kauan kuin laite saa virtaa. Puoliautomaattinen tila vaatii manuaalisen matkapuhelinverkkoon kytkeytymisen, mutta toimii muuten samoin kuin automaattinen tila. Projektin aikana kehitetyssä ohjelmistossa käytetään puoliautomaattista tilaa. Automaattisessa ja puoliautomaattisessa tilassa viestit lähetetään kutsumalla Particlen omaa Particle.publish()-funktiota. Manuaalisessa tilassa viestien lähettäminen vaatii tämän lisäksi vielä erillisen kutsun. Lisäksi laitteesta voidaan kytkeä rinnakkainen ajo eli threading päälle. Rinnakkainen ajo mahdollistaa

verkkoon yhdistämisen ja käyttötuntien laskemisen samaan aikaan.

Ohjelman toiminta alkaa setup-funktiosta. Funktiossa luetaan tietysta muistipaikasta mahdolliset aikaisemmin tallennetut käyttötunnit ja tallennetaan arvo käyttötuntilaskuriin. Jos käyttötunteja ei ole, alustetaan luku nollaksi. Tämän jälkeen siirrytään loop-funktioon. Listauksessa 5.1 on nähtävissä laitteen toimintalogiikka kyseisessä funktiossa. Loop-funktiossa odotetaan aluksi 60 000 millisekuntia, eli yksi minuutti, ja tämän jälkeen kasvatetaan käyttötuntilaskuria yhdellä. Käyttötunnit merkataan siis minuutin tarkkuudella.

Käyttötunnit kirjoitetaan myös muistiin, jolloin virran katketessa tiedot eivät häviä. On mahdollista, että virran katkeaminen juuri muistiinkirjoituksen aikana aiheuttaisi muistin korruptoitumista tai muun virhetilanteen, mutta kyseistä tilannetta ei saatu aiheutettua testien aikana. Riski arvioitiin pieneksi ja päätettiin jättää uudelleenarvioitavaksi pidempien testijaksojen jälkeen.

Kun käyttötuntilaskuri on jaollinen halutulla intervallilla, tarkastetaan, onko laite yhteydessä verkkoon. Jos yhteys verkkoon löytyy, lasketaan käyttötuntilaskurista käyttötuntien määrä ja lähetetään eteenpäin. Jos yhteyttä ei ole, aloitetaan verkkoon yhdistäminen. Verkkoon yhdistäminen voi kestää useita minuutteja. Seuraavalla lähetysintervallilla yritetään lähetystä uudelleen. Testauksen perusteella käyttötunnit halutaan lähettää web-palveluun 10 minuutin välein. Pidempiä lähetysintervalleja testattiin, mutta tällöin yhteys verkkoon saattaa kadota ennen kuin päästään uudestaan testaamaan, onko yhteyttä verkkoon ja voidaanko viestiä lähettää.

```
void loop() {
  // 1 minuutin odotus
    delay(60000);

  // Päivitetään käyttötunnit (minuutteina)
    updateUsageTime(1);

  // Jos kulunut kymmenen minuuttia, lähetetään viesti
  // pidempi intervalli hukkaa 2G-verkon
    if( minutes % 10 == 0){

        // Muunnetaan luku tunneiksi
        uint32_t hours = minutes / 60;

        // Particlen publish kutsu vaatii datan stringinä
        String data = String(hours);

        if (Particle.connected() == true) {
            connecting = false;
            // Lähetä data, kutsu pilvessä
            //"log_auth" nimistä webhookia
            Particle.publish("log_auth", data, PRIVATE);
        }
        else     // Jos ei ole yhteyttä verkkoon
        {
            if(!connecting){    // Jos ei olla yhdistämässä
                Particle.connect();      // Yhdistä verkkoon
                connecting = true;
            }
        }
    }
}

// Kasvata laskuria
void updateUsageTime(uint32_t step) {
    minutes = minutes + step ;  // kasvatetaan step:n verran
    EEPROM.put(addr, minutes);  // tallennetaan muistiin
}
```

Listaus 5.1 *Etälaitteen suurpiirteinen toimintalogiikka loop-funktiossa*

Kehityksen aikana Particle Electronissa huomattiin myös ongelma, joka oli laitteen meneminen kuuntelutilaan, jossa se ei saa tietoja SIM-kortilta. Laite ei toipunut tilasta ilman käyttäjän toimintaa. Verkkokeskustelujen perusteella ongelma oli yleinen ja se saadaan korjattua kuuntelemalla järjestelmän sisäisiä tapahtumia (system events), havaitsemalla kyseinen kuuntelutila tietystä sisäisestä tapahtumasta ja re-

setoimalla laite resetointikomennolla mahdollisimman pian tapahtuman jälkeen.

Ottamalla laitteen kuuntelutilaan meneminen huomioon, saatiin laitteelle toteutettua ohjelmisto, joka ei hukkaa kertyneitä käyttötunteja, eli on luotettava, ja joka selviytyy yleisimmistä mahdollisista ongelmatilanteista.

5.5 Testaus

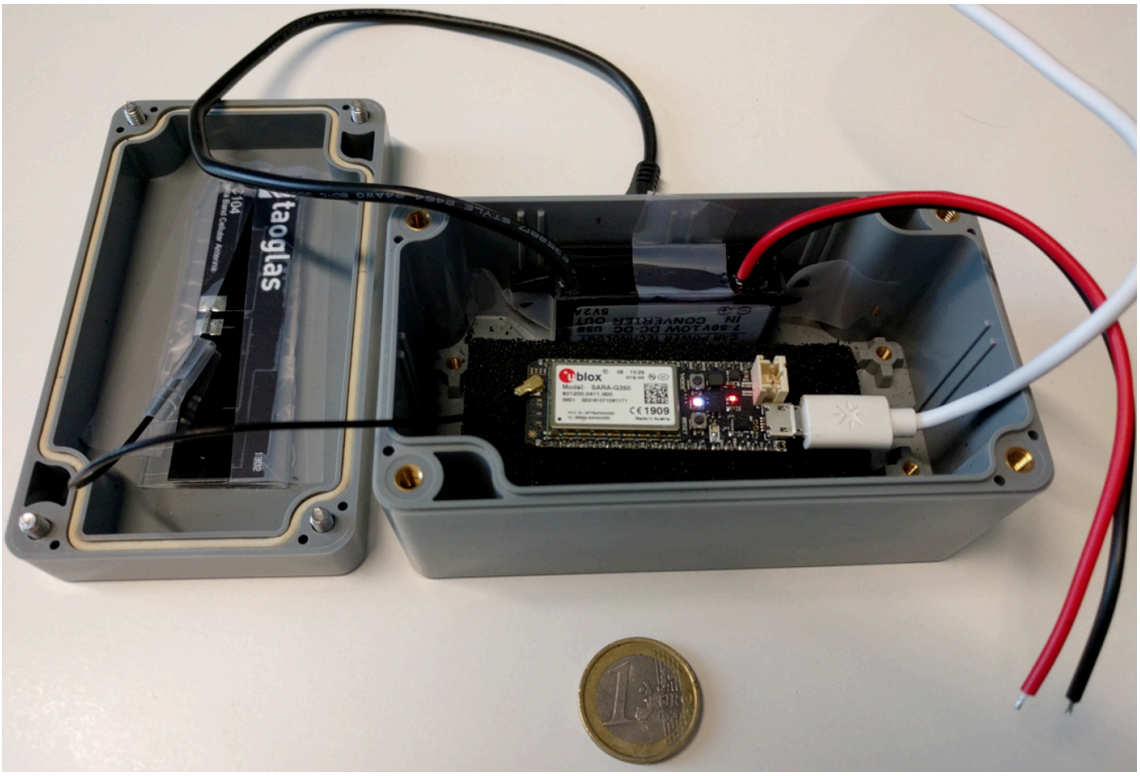
Etälaitetta testattiin toimisto-olosuhteissa useita päiviä. Tällä pyrittiin varmistamaan jännitemuuntajan toiminta, 2G-verkon kuuluvuuden riittävyys ja ohjelmiston robustisuus. Testien aikana käytettiin 12, 24 ja 48 voltin jännitteitä, jotka ovat työkoneiden sähköjärjestelmien yleisimpiä jännitteitä. Jännitemuuntimet eivät testauksen aikana osoittaneet epäluotettavuutta tai vääränlaista toimintaa.

Testauksen aikana huomattiin, että käyttötunnit täytyy lähettää useammin kuin tunnin välein. Laite ehti hukkaamaan verkon, jos se joutui odottamaan pitkiä aikoja verkkoon yhdistämisen ja tietojen lähettämisen välillä. Tästä syystä laitteen ohjelmistoa päivitettiin yrittämään viestin lähettämistä kymmenen minuutin välein tunnin sijaan.

Testaus osoitti että laite kokonaisuudessan on soveltuva käyttöönottoa varten. Laitteen ohjelmistosta löytyi virhe eli sattumanvarainen kuuntelutilaan meneminen, mutta se pystyttiin kiertämään. Viestien lähettämiselle löydettiin myös sopiva intervalli. Tällöin laite ei hukannu verkkoyhteyttä viestien lähettämisen välissä.

5.6 Käyttöönotto

Asiakas otti käyttöön kaksi testilaitetta. Testilaitteen prototyyppi on esitetty kuvassa 5.4. Ne kiinnitettiin trukkeihin, jotka olivat päivittäin aktiivisessa käytössä sisä- ja ulkotiloissa. Testijakso kesti yli kaksi kuukautta loppuvuodesta 2016. Kyseisen testijakson perusteella laitteet toimivat niille asetettujen vaatimusten mukaisesti. Jännitteenmuuntimet ovat siis osoittautuneet luotettaviksi eivätkä trukkien jännitteet ole rikkoneet niitä tai Particle Electronia. Verkkoyhteys on ollut laitteissa parempi kuin toimistotiloissa suoritetuissa testeissä. Jokin laitteen lähettämä viesti on voinut kadota esimerkiksi heikkojen yhteyksien takia, mutta suurin osa viesteistä on lokitiedostojen mukaan saapunut perille asti. Particle Electronin pilvipalvelu on myös osoittautunut luotettavaksi. Pilvipalvelulla ei ollut testijakson aikana ongelmia lähettää etälaitteilta saatuja viestejä eteenpäin.



Kuva 5.4 *Etälaitteen protoyyppi. Kyseisestä prototyypin kotelosta puuttuu vielä läpiviennit johdoille. Vieressä yhden euron kolikko.*

Selvittämättä testijakson aikana jäi se, kuinka tarkkoja etälaitteiden tuottamat lukemat ovat, kun niitä verrataan työkoneen käyttötuntimittariin. Tämä johtui käytännössä asiakkaan muista kiireistä.

6. JOHTOPÄÄTELMÄT JA JATKOKEHITYS

Luvussa esitetään työn lopputuloksesta syntyneet johtopäätelmät toteutetusta projektista. Näihin sisältyy valitun arkkitehtuurin, etälaitteen ja toteutuksen arviointia ja mahdollisten ongelmakohtien huomioimista. Luvussa kuvataan myös mahdollisesti kiinnostavat jatkokehityskohteet.

6.1 Johtopäätelmät

Valitut ratkaisut osoittautuivat hyväksi vaihtoehdoksi. Kaikki vaaditut ominaisuudet saatiin toteutettua. Valittujen ratkaisujen käytön oppiminen oli suhteellisen nopeaa ja intuitiivista tutoriaalien ja kokeneempien web-kehittäjien avulla. MEAN-arkkitehtuuri oli nopeasti opittava. Amazonin pilvipalvelut eivät myöskään rajoittaneet sovelluksen kehitystä.

Valituille ratkaisuille ei löytynyt ennustettavia ongelmakohtia. On mahdollista, että jonkin komponentin tuki tai kehitys lopetetaan, esimerkiksi joku Amazonin Web Servicesin pilvipalveluista, mutta tämäntyyppisiä riskejä ei ollut havaittavissa.

Kehitetty etälaite osoittautui luotettavaksi. Se oli kokonaisuutena melko edullinen ja koostui vain muutamasta erillisestä osasta. Particlen pilvipalvelu osoittautui luotettavaksi. Kaikki etälaitteiden lähettämät viestit tulivat perille testijakson aikana. Jos etälaitteeksi valitaan jossain vaiheessa jokin muu kuin Particlen tuote, jää myös Particlen pilvipalvelu pois järjestelmän arkkitehtuurista.

Suurimpana ongelmakohtana tai heikkoutena voidaan nähdä laitteen jatkokehitys. Koska siihen ei oltu integroitu muita antureita, kuten lämpötila- tai kiihtyvyysanturia, on ne liitettävä siihen jälkikäteen, mikä tarkoittaa melko paljon työtä. Laitteen ohjelmisto tulee päivittää, laitteisiin täytyy lisätä tarvittava anturi ja päivitetty ohjelmisto asentaa laitteisiin. Jos etälaitteen jo tarjoamat toiminnallisuus riittää myös tulevaisuudessa, eli käyttötuntien laskeminen laitteen ollessa päällä, sopii laite tähän tarkoitukseen hyvin.

Web-palvelun toteutuksesta tuli yksinkertainen. Sitä oli myös helppo laajentaa uusil-

la ominaisuuksilla. Sen kehitystä on mahdollista jatkaa luvussa 6.2 listatuista kehityskohteista. Web-palvelun toteutuksessa ei ole selviä ongelmakohtia. Jos palvelun käyttötarkoitus muuttuu radikaalisti, jatkokehitys ei välttämättä ole kannattavaa, vaan esimerkiksi valmiiden ja monipuolisten kaupallisten ratkaisujen käyttämistä kannattaa harkita. Kehitetty palvelu toimii siis hyvin käyttötarkoituksessa, johon se on alunperin luotu.

Kehitetystä palvelusta syntyi hyvin yleiskäyttöinen digitaalinen huoltokirja, joka ei ole riippuvainen etälaitteista ja niiden tarjoamasta käyttötuntidatasta. Palvelulla on siis asiakkaalle arvoa, vaikka etälaitteiden toiminta osoittautuisikin epävarmaksi, tai niiden tarjoama data hyödyttömäksi. Asiakkaalla on myös mahdollisuus myydä samaa palvelua käyttöön muille yrityksille ja näin hyötyä tehdystä investoinnista. Palvelu ei myöskään ole riippuvainen tietystä etälaitteesta, vaan mikä tahansa samassa muodossa dataa käsittelevä laite pystyy HTTP-yhteyden välityksellä toimimaan toteutetun palvelun kanssa.

6.2 Jatkokehitys

Jatkokehityskohteita palvelusta löytyy useita. Kohteet ovat uusia ominaisuuksia ja vanhojen ominaisuuksien päivittämistä. Jatkokehityskohteiden toteuttamisella palvelua pystytään rakentamaan käyttäjäystävällisemmäksi, tietoturvallisemmaksi ja enemmän arvoa tuottavaksi niin asiakkaille kuin huoltoyrityksillekin.

6.2.1 Toteutuksen arviointi ja datan luotettavuus

Toteutettua palvelua ja valittua etälaitetta olisi hyvä arvioida pidemmän testijakson jälkeen. Arviointi kannattaa suorittaa haastattelemalla järjestelmän käyttäjiä eli asiakkaita, huoltomiehiä ja ylläpitäjiä. Arvioinnin kohteeksi olisi hyvä ottaa ainakin käyttöliittymän ulkonäkö ja intuitiivisuus. Samalla olisi hyvä selvittää eri ominaisuuksien käyttöasteet, mitä ominaisuutta käytetään paljon ja onko ominaisuuksia, joita ei käytetä. Samalla voidaan huomata tarpeita, johon nykyinen järjestelmä ei kykene.

Kerätyn datan tarkkuus on tällä hetkellä epäselvä. Testidataa ei ole vielä useasta koneesta, eikä tietoa siitä, kuinka tarkasti etälaitteen ilmoittamat käyttötunnit vastaavat työkoneen omia lukemia. Kyseinen näkökulma olisikin hyvä testata laajemmin, jotta etälaitteen luotettavuudesta voidaan varmistua. Testaus olisi hyvä suorittaa ennen kuin etälaitteita otetaan laajasti käyttöön. Kerätyn datan tarkkuus olisi hyödyllistä testata useiden työkonetyyppien kanssa.

6.2.2 Käyttäjätunnusten hallinnoinnin päivitys

Tämänhetkinen versio web-palvelusta ei salli asiakas- tai huoltomieskäyttäjien asettaa itse salasanaansa tai vaihtaa sitä. Admin-käyttäjän täytyy asettaa salasana tilin luonnin yhteydessä. Koska asiakas tai huoltomies ei voi tallentaa palveluun mitään henkilötietoja, niihin liittyviä henkilötietoturvariskejä ominaisuus ei aiheuta. Muita tietoturvariskejä tässä työssä ei arvioitu. Kyseessä ei kuitenkaan ole vahvasti suojattu järjestelmä, joten järjestelmään kohdistuu varmasti tietoturvauhkia. Palvelun käyttömukavuus kuitenkin paranisi, jos käyttäjä voisi itse asettaa salasanansa ja vaihtaa sitä.

6.2.3 Ulkonäkö- ja käytettävyyspäivitys

Palvelu on ulkonäöltään melko geneerinen, koska se käyttää pääasiassa valmiita kirjastoja. Palvelun ulkonäön päivittäminen parantaisi käytettävyyttä ja auttaisi palvelun mainostamisessa sekä brändin luomisessa. Palvelua ei ole optimoitu mobiililaitteille, mikä voisi joissain käyttötarkoituksissa olla hyödyllistä. Huoltomies saattaa tulevaisuudessa kuljettaa mukanaan vain mobiililaitetta käydessään huoltamassa työkoneita. Tällöin mobiililaitteille optimioitu web-palvelu tehostaisi töiden kirjaamista palveluun.

6.2.4 Palvelun muokkaaminen myytäväksi palveluksi

Palvelua ei tällä hetkellä ole rakennettu useiden huoltoyritysten samanaikaiseen käyttöön. Tällä hetkellä se on mahdollista vain luomalla palvelusta kopio toiselle palvelimelle. Kehittämällä palvelulle korkeamman tason ylläpitäjänäkymä, joka voisi luoda useita huoltoyrityksiä palveluun, palvelua voisi myydä muille huoltoyrityksille käytettäväksi.

6.2.5 Datan visualisointi

Palveluun saapuvaa dataa ei koota yhteen tai visualisoida millään tavalla. Kertyneet tunnit ilmoitetaan konekohtaisesti numeroin. Visualisoinnilla pystyisi havaitsemaan kaikkien työkoneiden lähettämässä datassa mahdollisesti toistuvia ilmiöitä, esimerkiksi käyttötuntien vaihtelua vuodenaikojen mukaan tai tiettyjen kuukausien aikana. Tällaisia visualisointeja voisivat olla muun muassa käyttötuntien kertyminen tietyltä työkoneelta per päivä kuukauden ajalta tai tietyn asiakkaan kaikkien koneiden kertyneet käyttötunnit per viikko useamman vuoden ajalta.

6.2.6 Lisädatan kerääminen

Eräs dataan liittyvä jatkokehityskohde on lisädatan kerääminen. Ominaisuus vaatii etälaitteelle lisää antureita, esimerkiksi kiihtyvyysanturin tai lämpötila-anturin. Riippuen anturista, on hyvä tutkia kannattaako vanhoihin etälaitteisiin kiinnittää uusia antureita, vai etsiä uusi etälaite, joka sopii tarkoitukseen paremmin.

Kiihtyvyysanturin avulla voitaisiin havaita, milloin laite on niin sanotusti tyhjäkäynnillä. Tyhjäkäyntiä halutaan välttää ja tyhjäkäynnin sijaan työkone olisikin parempi sammuttaa, jottei polttoainetta kulu turhaan. Mikrofonin avulla voitaisiin kuunnella työkoneen moottorin käyntiääntä ja havaita siinä muutoksia. Muutos käyntiäänessä saattaisi tarkoittaa jonkun osan selkeää kulumista tai jopa rikkoutumista. Näin etälaite voisi varoittaa rikkoutuneesta osasta, kehottaa sammuttamaan työkone ja tutkimaan käyntiäänen muutoksen lähdettä.

GPS-moduulin avulla laitteen nopeutta ja sijaintia voitaisiin tarkkailla. Tällöin olisi mahdollista tutkia, miten ja kuinka nopeasti laite liikkuu ja missä se tietyllä ajanhetkellä on. GPS-seurannan avulla voitaisiin myös helpottaa työkoneen tyhjäkäynti havaitsemista. Lämpötila-anturin avulla voitaisiin havaita moottorin ylikuumenemista, tai muiden oleellisten osien lämpötiloja. Sensorivaihtoehtoja on siis useita, joten niistä tulisi valita ensimmäiseksi kehityskohteeksi tärkein ominaisuus.

6.2.7 Datan analysointi

Yksi jatkokehitysmahdollisuus on analysoida kerättyä dataa olettaen, että sitä on saatu kerättyä riittävästi. Erilaiset koneoppimisen metodit mahdollistavat monimutkaisetkin data-analyysit kerätystä datasta. Koneoppimisen avulla datasta voidaan löytää piileviä säännöllisyyksiä tai opettaa ohjelmisto tunnistamaan datasta kiinnostavia asioita.

Kerätyn käyttötuntidatan avulla pystyttäisiin esimerkiksi ennustamaan tulevien huoltojen päivämääriä yksinkertaisella lineaarisella regressiolla tai tunnistamaan huonokuntoisempia työkoneita koneelle tehtyjen korjausten frekvenssin perusteella. Jos dataa kertyy paljon, erityyppisten asiakkaiden luokittelu saattaisi olla mahdollista sen perusteella, kuinka paljon työkoneita on ja miten niitä käytetään. Datan analysoinnin mahdollisuudet kasvavat, jos etälaitteissa on useampia antureita, joiden avulla dataa kerätään. Tällöin dataa kertyy suurempi määrä ja piilevien säännöllisyyksien löytäminen datasta helpottuu.

LÄHTEET

[1] “About javascript,” verkkosivu (Viitattu 14.11.2016): https://developer.mozilla.org/en-US/docs/Web/JavaScript/About_JavaScript.

[2] “Amazon ec2,” verkkosivu (Viitattu 12.12.2016): https://aws.amazon.com/ec2/.

[3] “Amazon echo,” verkkosivu (Viitattu 30.12.2016): https://en.wikipedia.org/wiki/Amazon_Echo.

[4] “Amazon ses,” verkkosivu (Viitattu 12.12.2016): https://aws.amazon.com/ses/?nc2=h_m1.

[5] “Angularjs,” verkkosivu (Viitattu 1.12.2016): https://angularjs.org/.

[6] “Arduino,” verkkosivu (Viitattu 13.12.2016): https://en.wikipedia.org/wiki/Arduino.

[7] “Aws lambda,” verkkosivu (Viitattu 12.12.2016): https://aws.amazon.com/lambda/?nc2=h_m1.

[8] “Chakracore,” verkkosivu (Viitattu 14.11.2016): https://github.com/Microsoft/ChakraCore.

[9] “Developer survey results 2016,” verkkosivu (Viitattu 14.11.2016): http://stackoverflow.com/research/developer-survey-2016.

[10] “Express.js,” verkkosivu (Viitattu 1.12.2016): http://techforgeek.com/expressjs/.

[11] “Express.js-kotisivu,” verkkosivu (Viitattu 1.12.2016): http://expressjs.com/.

[12] “Introducing json,” verkkosivu (Viitattu 8.11.2016): http://json.org/.

[13] “Mode switching,” verkkosivu (Viitattu 4.1.2017): https://docs.particle.io/support/troubleshooting/mode-switching/electron/.

[14] “Mongodb,” verkkosivu (Viitattu 5.12.2016): https://www.mongodb.com/.

[15] “Mongodb manual: Documents,” verkkosivu (Viitattu 5.12.2016): https://docs.mongodb.com/manual/core/document/.

[16] “Node.js,” verkkosivu (Viitattu 21.11.2016): https://nodejs.org/en/.

[17] "Nosql," verkkosivu (Viitattu 21.11.2016): https://en.wikipedia.org/wiki/NoSQL.

[18] "Particle cloud," verkkosivu (Viitattu 19.12.2016): https://www.particle.io/products/platform/particle-cloud.

[19] "Particle electron," verkkosivu (Viitattu 13.12.2016): https://store.particle.io/collections/electron.

[20] "Raspberry pi," verkkosivu (Viitattu 13.12.2016): https://en.wikipedia.org/wiki/Raspberry_Pi.

[21] "Relational database," verkkosivu (Viitattu 21.11.2016): https://en.wikipedia.org/wiki/Relational_database.

[22] "Representational state transfer," verkkosivu (Viitattu 30.11.2016): https://en.wikipedia.org/wiki/Representational_state_transfer.

[23] "Rfid," verkkosivu (Viitattu 30.12.2016): https://fi.wikipedia.org/wiki/RFID.

[24] "Thingsee one lite," verkkosivu (Viitattu 13.12.2016): https://thingsee.com/thingsee-one-lite.

[25] "u-blox c027," verkkosivu (Viitattu 13.12.2016): https://www.u-blox.com/en/product/c027.

[26] "u-blox c027," verkkosivu (Viitattu 16.12.2016): https://developer.mbed.org/platforms/u-blox-C027/.

[27] "Using http methods for restful services," verkkosivu (Viitattu 8.11.2016): http://www.restapitutorial.com/lessons/httpmethods.html.

[28] "Web application," verkkosivu (Viitattu 1.12.2016): https://en.wikipedia.org/wiki/Web_application.

[29] "Webhooks," verkkosivu (Viitattu 19.12.2016): https://docs.particle.io/guide/tools-and-features/webhooks/.

[30] F. Buschmann, K. Henney, and D. Schimdt, *Pattern-oriented Software Architecture: On Patterns and Pattern Language*. John Wiley & Sons, 2007, vol. 5.

[31] E. F. Codd, "A relational model of data for large shared data banks," *Commun. ACM*, vol. 13, no. 6, pp. 377–387, Jun. 1970. [Online]. Available: http://doi.acm.org/10.1145/362384.362685

[32] *The JSON Data Interchange Format*, ECMA International, 2013, 1st Edition Saatavilla: http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf.

[33] *Standard ECMA-262: ECMAScriptő 2016 Language Specification*, ECMA International, 2016, 7st Edition Saatavilla (Viitattu 14.11.2016): http://www.ecma-international.org/ecma-262/7.0/index.html.

[34] R. T. Fielding, "Architectural styles and the design of network-based software architectures," Ph.D. dissertation, University of California, Irvine, 2000.

[35] T. Haerder and A. Reuter, "Principles of transaction-oriented database recovery," *ACM Comput. Surv.*, vol. 15, no. 4, pp. 287–317, Dec. 1983. [Online]. Available: http://doi.acm.org/10.1145/289.291

[36] G. Kortuem, F. Kawsar, V. Sundramoorthy, and D. Fitton, "Smart objects as building blocks for the internet of things," *IEEE Internet Computing*, vol. 14, no. 1, pp. 44–51, Jan 2010.

[37] P. Krill, "Javascript creator ponders past, future," verkkosivu (Viitattu 14.11.2016): http://www.infoworld.com/article/2653798/application-development/javascript-creator-ponders-past--future.html, 2008.

[38] A. Leff and J. T. Rayfield, "Web-application development using the model/-view/controller design pattern," in *Proceedings Fifth IEEE International Enterprise Distributed Object Computing Conference*, 2001, pp. 118–127.

[39] P. Louridas, "Component stacks for enterprise applications," *IEEE Software*, vol. 33, no. 2, pp. 93–98, Mar 2016.

[40] A. B. M. Moniruzzaman and S. A. Hossain, "Nosql database: New era of databases for big data analytics - classification, characteristics and comparison," *CoRR*, vol. abs/1307.0191, 2013. [Online]. Available: http://arxiv.org/abs/1307.0191

[41] K. Panetta, "7 technologies underpin the hype cycle for the internet of things, 2016," verkkosivu (Viitattu 30.12.2016): http://www.gartner.com/smarterwithgartner/7-technologies-underpin-the-hype-cycle-for-the-internet-of-things-2016/, Nov 2016.

[42] A. J. Poulter, S. J. Johnston, and S. J. Cox, "Using the mean stack to implement a restful service for an internet of things application," in *2015 IEEE 2nd World Forum on Internet of Things (WF-IoT)*, Dec 2015, pp. 280–285.

[43] S. Srinivasan, *Cloud computing basics*. Springer, 2014.